공학박사 학위 논문

영역 분할을 활용한 Hardware 기반 고속 Local Feature Matching

Hardware-based Fast Local Feature Matching
Using Region Segmentation

2014년 2월

서울대학교 대학원
전기·정보공학부
장 정 환

영역 분할을 활용한 Hardware 기반 고속 Local Feature Matching

Hardware-based Fast Local Feature Matching Using Region Segmentation

지도교수 김 수 환

이 논문을 공학박사 학위논문으로 제출함

2014년 2월

서울대학교 대학원

전기·정보공학부

장 정 환

장정환의 공학박사 학위 논문을 인준함

2014년 2월

위 원 장: 채 수 익 (인)

부위원장: 김 수 환 (인)

위 원: 최 진 영 (인)

위 원: 이 혁 재 (인)

위 원: 김 진 성 (인)

초 록

최근 디지털 저장 장치와 개인 영역에서도 surveillance system 이 빠르게 늘어남에 따라서, 저장된 이미지 데이터의 양이 급격하게 늘어나고 있으며, 저장된 데이터 내에서 유사 이미지 검색, 분류, 인식 등의 응용이 점차 증가하고 있다. 이와 같은 영상 인식 시스템에서 SIFT local feature 를 이용한 영상 인식 방법은 다양한 영상의 변화에 강인한 특성을 가지기 때문에, 최근 들어 모바일, robot navigation, object recognition 등 여러 분야에서 활용도가 점차 높아지고 있다. 영상 인식 시스템은 많은 양의 연산을 필요로 하는 단계들을 거쳐 수행되기 때문에, mobile 과 같은 제안된 resource 를 가지는 환경에서 고속으로 처리하는데 어려움이 있다. 본 연구에서는 고속으로 영상인식을 수행하기 위한 방법들을 제안한다.

SIFT 하드웨어 에서의 속도 향상 방법으로 adaptive SIFT keypoint 생성방법을 제안하였다. Keypoint 의 수는 전체 하드웨어의 수행시간에 영향을 미치는 요소이다. 이를 조절하기 위해 본 연구에서는 블록단위로 영상의 복잡도를 예측하고, 블록의 contrast threshold 값을 달리 적용하여, keypoint 의 수를 조절하는 방법을 사용하였다. 제안한 방법은 SIFT 하드웨어의 pipeline 구조 유지가 가능하고, keypoint 발생 분포면에서 유리한 성능을 보인다. 또한 이를 통해 전체 수행시간을 단축할수 있음을 확인하였다.

SIFT 에서 생성된 descriptor 사이에 Euclidean distance 의 유사성을 사용한 matching 방법은 많은 수의 잘못된 correspondence 를 포함하게 된다.

초기의 correspondence 로부터 inlier 와 outlier 를 구분하기 위한 clustering 기반의 feature matching 방법은 이미지내에 모든 correspondence 를 대상으로 clustering 을 수행하므로, 많은 수의 연산이 필요하다. 본 연구에서는 영역 단위로 clustering 을 수행하는 방법을 제안하고, 영역내의 clustering 결과의 집합으로 전체의 결과를 표시하였다. Clustering 을 수행하는 영역은 이미지의 segmentation 정보를 사용하여, 유사한 특성의 correspondence 들을 포함하는 인접 segment 들을 합하여 구성하였다. 제안한 방법은 동등 수준의 성능에서도 수행시간을 크게 단축할수 있음을 확인하였다.

Watershed segmentation 방법은 이미지내의 값들을 순차적으로 사용하기 때문에 병렬처리가 어렵다. 본 연구에서는 이미지를 블록 단위로 나누고, 블록간 독립적인 연산을 가능하게 하는 방법을 제안하였다. 블록 단위 연산에서 상호의존적인 부분은 블록 경계면에서의 gradient 값과 인접해 있는 segment 들간의 root 값을 통해 처리하였다. 또한, 하나의 segment 로 예측되는 블록들에 대해선 watershed 의 일부 연산을 수행하지 않도록 하였다. 제안된 방법은 기존 watershed 의 결과와 동일한 수의 segment 를 가지는 것을 확인하였고, 블록별 병렬처리를 통해 수행시간이 줄어드는 것을 확인하였다.

주요어: SIFT, FAST, 하드웨어, SoC, Feature matching, Segmentation
학번: 2008-30240

차 례

그림 목차

표 목차

제1장 서론

1.1 연구 배경

최근 디지털 저장 장치와 CCTV 의 증가로 인하여 저장된 이미지 데이터의 양이 급격하게 증가하고 있으며, 저장된 데이터 내에서 유사 이미지 검색, 분류, 인식 등의 응용이 점차 증가하고 있다.

여러 영상 인식 시스템 중 local feature 를 이용한 영상 인식 방법들은 noise 나 light variation, viewpoint 변화 등과 같은 영상 변화에 강인한 특성을 가지고 있기 때문에, 최근 들어 컴퓨터 비젼 분야에서 많이 연구되고 있다[3][7].

Local feature 기반의 영상 인식 시스템은 그림 1.1 과 같이 구성할 수 있다. Local feature 를 이용한 영상 인식 과정은 feature extraction 과정과 feature matching 과정으로 구성되어 있다.

Feature extraction 단계에서는 영상에서 주목할 만한 feature point 들을 추출하여 feature 주변의 영상을 기술하는 feature descriptor 로 만드는 과정까지를 포함한다. 이 단계 이후의 연산은 입력 이미지가 아닌, feature descriptor 를 통해 이루어진다. 이 단계에서는, 필요한 정보만을 추출하기 위해 수많은 입력 픽셀에 대한 연산을 수행해야 하기 때문에 매우 많은 연산량이 필요하다. 또한, 이 단계는, 필요한 정보만을 남기고 나머지는 버리기 때문에 그 결과에 따라 물체 인식 결과가 크게 바뀔 수 있다.

Feature matching 단계에서는 feature point 들간에 상관성을 조사하여 feature 들간에 대응 관계를 찾는 과정을 수행한다[4]. 대응 관계를 찾는 과정은 feature extraction 통해서 계산된 feature descriptor 간에 유사성을 통해 판단한다.

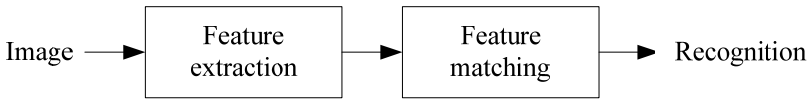


그림 1.1 General recognition system

특히 최근 들어 많은 양의 연산을 필요로 하는 영상 처리 방법들을 모바일에서 수행하기 시작하면서, 위와 같은 인식 시스템에 대한 알고리즘을 효과적으로 구현하여 속도를 높이는 것이 중요해 지고 있다. 모바일 환경에서 주로 이루어지는 영상 처리 방법들로는 영상 검색, 얼굴 인식, 파노라마 이미지 생성 등이 있다. 또한 최근 검색 엔진 등에서도 영상 검색 기술이 적용되고 있다[1][2][5][6][8]. 따라서, 이러한 local feature 기반의 recognition system 을 실시간으로 처리하기 위한 방법들에 대한 연구가 많이 진행되고 있다.

본 연구에서는 local feature 기반의 recognition system 을 수행하기 위해서 단계별로 각각의 특성을 분석하고, 효과적으로 연산하기 위한 방법들을 제안한다.

1.2 연구 내용

본 연구에서는 효율적으로 물체를 인식하기 위한 방법들을 연구하였다. 이 중 SIFT 는 여러 가지 방법 중, 가장 뛰어난 성능을 보이는 feature extraction 방법으로 알려져 있다[12][13]. 하지만 SIFT 는 많은 연산량을 필요로 하기 때문에 실시간 처리를 위한 방법으로는 많은 어려움이 있다. 이러한 문제점을 개선하고자, 본 연구에서는 하드웨어로 구현된 keypoint (feature point) 와 descriptor generation 과정을 이용하였고, 많은 수의 keypoint 로 인해서 길어진 수행 시간을 줄이기 위해서, 하드웨어에 적합한 블록 기반의 keypoint 수 조절 방법을 제안하였다. 이와 같은 방법들을 이용하여, 본 연구에서는 전체 하드웨어 속도를 향상하는 방법을 제안하였다.

Descriptor 를 사용해서, local patch 의 유사성만으로 상관성을 추출하면, 많은 수의 잘못된 correspondence 가 발생하게 된다. 따라서, correspondence 가운데, correct correspondence(inlier) 와 wrong correspondence(outlier) 를 구분할 필요가 있다. 본 연구에서는 다양한 영상 변형에서도 효과적으로 inlier 와 outlier 를 구분하는 clustering 기반의 matching 방법을 사용하였다. 그러나 clustering 기반의 알고리즘은 correspondence 의 수에 따라서 급격하게 처리할 연산이 늘어나는 문제점을 가지고 있다. 따라서 본 연구에서는 clustering 이 될 가능성이 높은 correspondence 들을 region 단위로 나누어, region 내에서 clustering 을 수행하는 방법을 제안하였다.

또한 region 단위로 clustering 을 수행하기 위해서 필요한 segmentation 을 효과적으로 수행하기 위해서, 블록 기반의 watershed 방법을 제안하였다.

제안된 방법의 검증은 그림 1.2 와 같이 하드웨어상에서 keypoint 수 조절 방법을 적용하여 이에 대한 성능 향상을 확인하였고, 소프트웨어 구현을 통해 region 기반의 clustering 알고리즘과 segmentation 알고리즘의 성능 검증을 수행하였다.

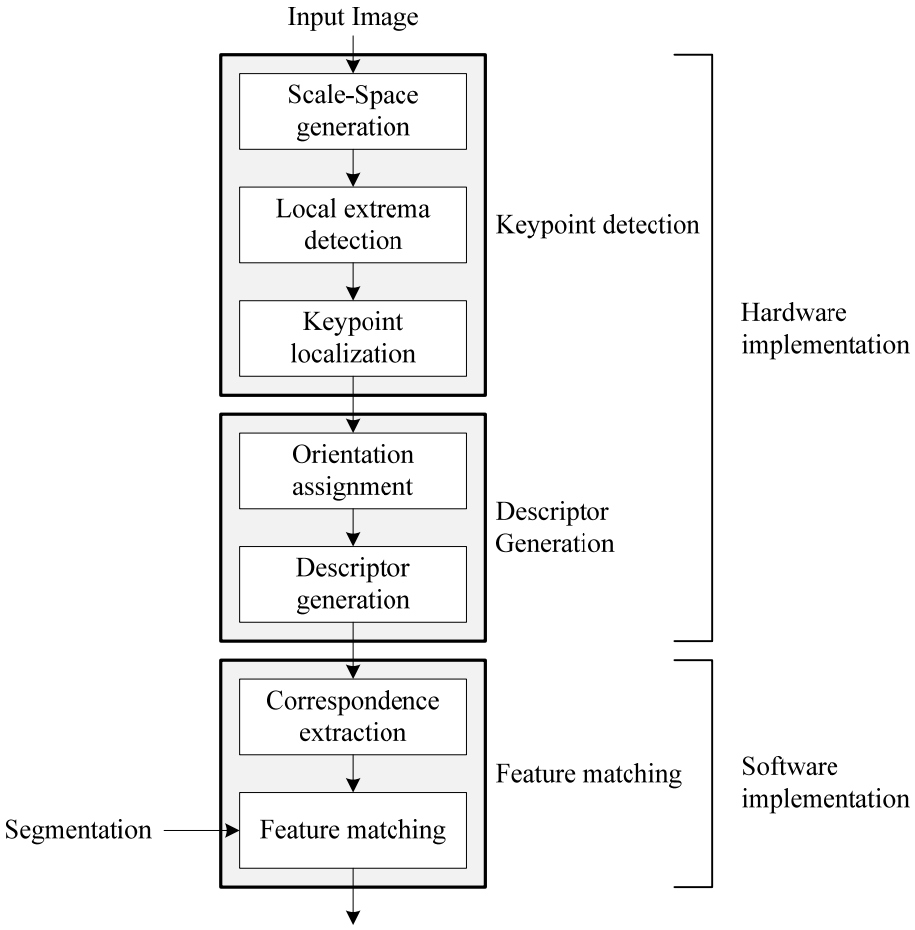


그림 1.2 시스템 구성 및 검증 flow

1.3 논문 구성

본 연구의 나머지 구성은 다음과 같다. 2장에서는 본 연구의 기본이 되는 이론과 기존 연구를 소개 한다. 3장에서 adaptive keypoint generation 을 위한 하드웨어 구조를 제안하며, 4장에서는 생성된 keypoint 로부터 inlier 와 outlier 를 구분하는 방법으로 사용하는 clustering 을 효과적으로 구현하는 region constrained feature matching 방법을 제안한다. 5장에서는 4장에서의 region 을 구성하는 단위로서 사용되는 watershed segmentation 을 병렬 처리가 가능한 블록 단위로 수행하는 방법을 제안한다. 마지막으로 6장에서 결론을 맺는다.

제2장 관련 연구

이 장에서는 본 연구의 배경 이론인, SIFT 의 keypoint 와 descriptor 생성 알고리즘에 대해 설명하고, 이에 대한 문제점을 설명한다. 또한 feature matching 을 위한 기존 연구와 그에 대한 문제점을 설명한다.

2.1 SIFT (Scale-Invariant Feature Transform)

Feature detector 는 이미지에서 corner, blob, edge 와 같은 이미지 내의 특징된 부분을 추출하는 방법을 의미한다. Feature detector 들에서는 이러한 특징된 부분들을 표현하기 위해서 특징의 위치를 표현하는 keypoint 와 그 위치에서의 주변 영상의 특징을 표현하는 descriptor 을 사용한다.

이중 SIFT 는 Lowe[9] 가 제안한 local feature 의 한 종류로, 여러 local feature 중 가장 성능이 우수한 방법으로 알려져 있다. SIFT 는 크게 keypoint detector 와 feature 주변의 영상을 기술하는 descriptor 를 생성하는 부분으로 구성되어 있다. 또한 [9] 논문에서는 SIFT descriptor 간 유사성을 비교하여 correspondence 를 생성하는 과정까지를 포함하고 있다. 그림 1.2 와 같이 keypoint detection 단계에서는 scale-space generation, local extrema detection, keypoint localization 의 세부적인 연산으로 구성되어 있으며, descriptor 생성 단계에서는 orientation assignment, descriptor generation 의 세부적인 연산으로 구성되어 있다. 이후 절에서는 각각의 세

부 부분에 대해서 설명한다.

2.1.1 Scale-space generation

SIFT 에서는 scale 변화에 따른 keypoint 를 추출하고, 이를 통해, scale 이 변화된 영상에서도 이미지간 matching 이 가능하도록 하고 있다. 이를 위해 이미지의 scale-space 를 생성하고, 여러 scale 에서 keypoint 를 생성하게 된다[10].

Scale-space generation 단계에서는 입력 이미지의 Gaussian filtering 과정을 통해서, Gaussian filtering 된 이미지를 생성하고, 이를 통해, DoG(Difference Of Gaussian) 이미지를 생성한다.

먼저 입력 이미지 $I(x,y)$ 로부터 Gaussian filtering 된 이미지, $L_i(x,y)$ 는 다음과 같은 식을 통해 계산된다.

$$L_i(x,y) = G(x,y,\sigma_i) * I(x,y) \quad for\ i = 0,1,\dots,S+2 \qquad (2.1)$$

$$G(x,y,\sigma_i) = \frac{1}{2\pi\sigma_i^2} e^{-\frac{x^2+y^2}{2\sigma_i^2}} \quad where\ \sigma_i = \sigma_0 \cdot 2^{\frac{i}{S}} \qquad (2.2)$$

입력 이미지에 대한 Gaussian filtering 된 이미지는 식 (2.1)와 같이 입력 이미지와 Gaussian filter 의 convolution 과정을 통해서 계산된다. 이때, 사용되는 Gaussian kernel 은 식 (2.2)와 같이 나타난다.

Gaussian filtering 된 이미지는 Gaussian kernel 에 따라서 다르게 생성 되는데, Gaussian kernel 의 scale σ_i 로 생성된 filtering 된 이미지를 $L_i(x,y)$ 로 표시한다. Octave 는 scale-space 를 구성하기 위해서 down-sampling

하고 나서, 다음 down-sampling 까지의 Gaussian filtering 된 이미지들의 집합으로 구성된다. 식 (2.2) 에서 S 는 octave 당 scale 의 개수이다. 본 연구에서는 S 값으로 3 을 사용하였다. Octave 당 3개의 scale 에 대해서, 각각 feature 를 생성하기 위해서는 S+2 의 DoG 이미지가 필요하게 된다. 또한 DoG 이미지는 S+3 개의 Gaussian filtering 된 이미지로부터 만들어진다. 따라서, S 가 3 일 경우 6 개의 Gaussian filtering 된 이미지를 만들어야 한다.

예를 들어 그림 2.1 과 같이 3x3 크기의 Gaussian filter 를 사용하여, 1 픽셀의 결과를 생성하기 위해서, 3x3 크기의 입력 영상과 3x3 크기의 Gaussian filter 의 convolution 연산을 수행하게 된다[21].

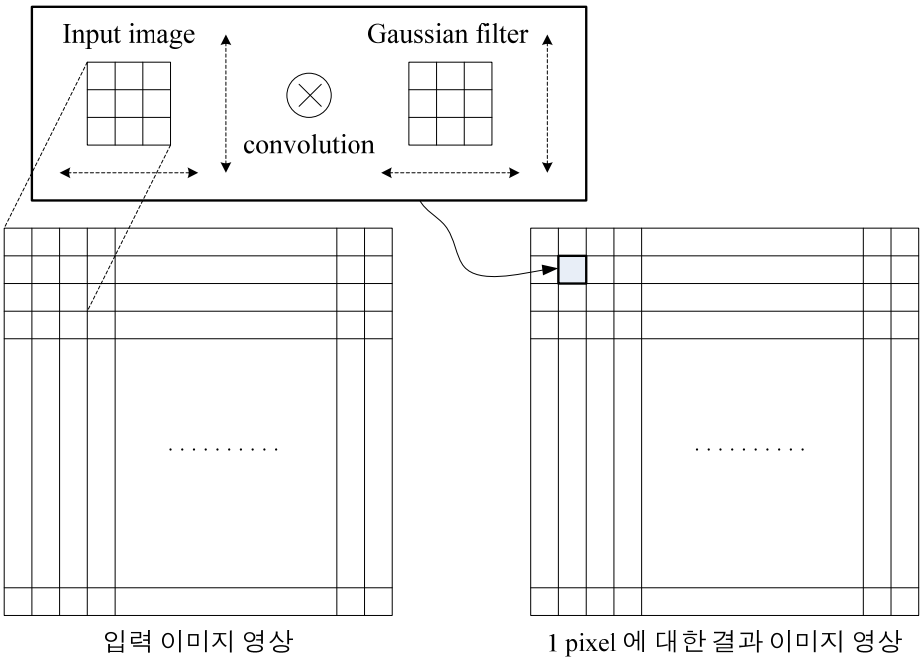


그림 2.1 Gaussian filtering 된 이미지 생성을 위한 연산 과정

DoG 이미지는 두 개의 Gaussian filtering 된 이미지의 차 영상을 의미한다. DoG 이미지는 식 (2.3) 을 통해 계산이 가능하다.

$$D_i(x,y) = \big(G(x,y,\sigma_{i+1}) - G(x,y,\sigma_i)\big) * I(x,y) \\ = L_{i+1}(x,y) - L_i(x,y) \tag{2.3}$$

표 2.1 는 S 가 3 일 경우 Octave 0 에 대해서 Gaussian filter 의 scale 에 따른 Gaussian filtering 된 이미지와 DoG 이미지간의 관계를 보여준다. S 가 3 일 경우 5개의 DoG 이미지가 생성됨을 볼 수 있다.

표 2.1 Scale 에 따른 filtered image 와 DoG 영상 생성

Octave 0						
Filter scale	σ_0	$\sigma_0 \cdot 2^{1/3}$	$\sigma_0 \cdot 2^{2/3}$	$\sigma_0 \cdot 2$	$\sigma_0 \cdot 2^{4/3}$	$\sigma_0 \cdot 2^{5/3}$
Filtering 된 이미지	L_0	L_1	L_2	L_3	L_4	L_5
DoG image		D_0	D_1	D_2	D_3	D_4

그림 2.2 는 Oxford dataset 의 Graffiti 이미지를 사용하였을 때, Octave 0,1,2 에서 생성된 Gaussian pyramid 이미지를 보여준다. Octave 1 은 원본 영상을 $2\sigma_0$ 의 Gaussian filter scale 로 filtering 한 L_3 의 영상을 가로 세로 방향으로 각각 1/2 씩 down-sampling 하여, 두 번째 octave 의 L_0 영상을 만든다.

이때 적용되는 Gaussian filter 는 그림 2.2 의 상단 이미지와 같이 octave 내에서 filter scale 에 따라 다른 크기의 Gaussian filter 가 사용된다. 그림

2.3 은 Octave 0 에서 Gaussian filtering 된 이미지로부터 생성된 DoG 를 보여준다. 그림에서와 같이 DoG 이미지는 서로 다른 Gaussian filter 의 scale 로 생성된 이미지의 차이기 때문에, edge 나 blob 이 존재하는 부분에서 local extrema, 즉 local minimum 이나 local maximum 이 발생한다. 그림에서의 DoG 이미지는 시각적으로 강조해서 보여주기 위해 histogram equalization 과정을 수행하여, 이미지의 contrast 를 증가시킨 이미지이다.

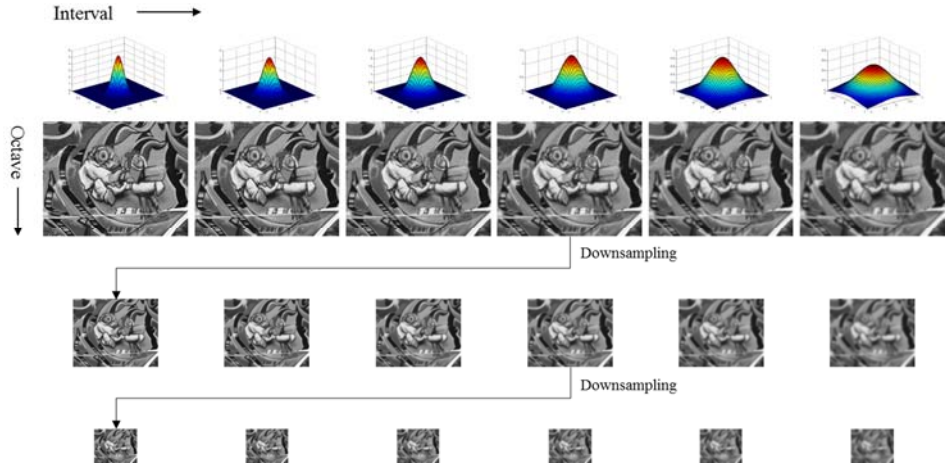


그림 2.2 Gaussian pyramid images

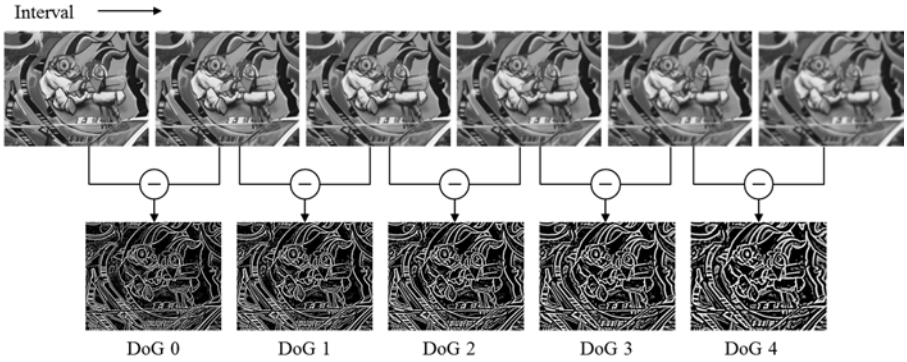


그림 2.3 Octave 0 DoG images

2.1.2 Local extrema detection

이미지의 변화에 대해 invariant 한 keypoint를 추출하기 위해서, local extrema detector 단계에서는 그림 2.4 과 같이 인접한 3개의 DoG 이미지를 사용한다. 현재 픽셀이 속한 DoG 이미지와 인접한 2 개의 DoG 이미지를 사용하여 구성한 3x3x3 형태의 픽셀 집합들에서 상하 좌우로 인접한 총 26개의 픽셀을 비교한다. 따라서, 5개의 DoG 이미지로부터 3 개의 keypoint detection 를 위한 pyramid 가 구성가능하다. 이때 현재 픽셀이 local minimum 이나 local maximum 인 경우 특징점을 위한 후보로 추출된다.

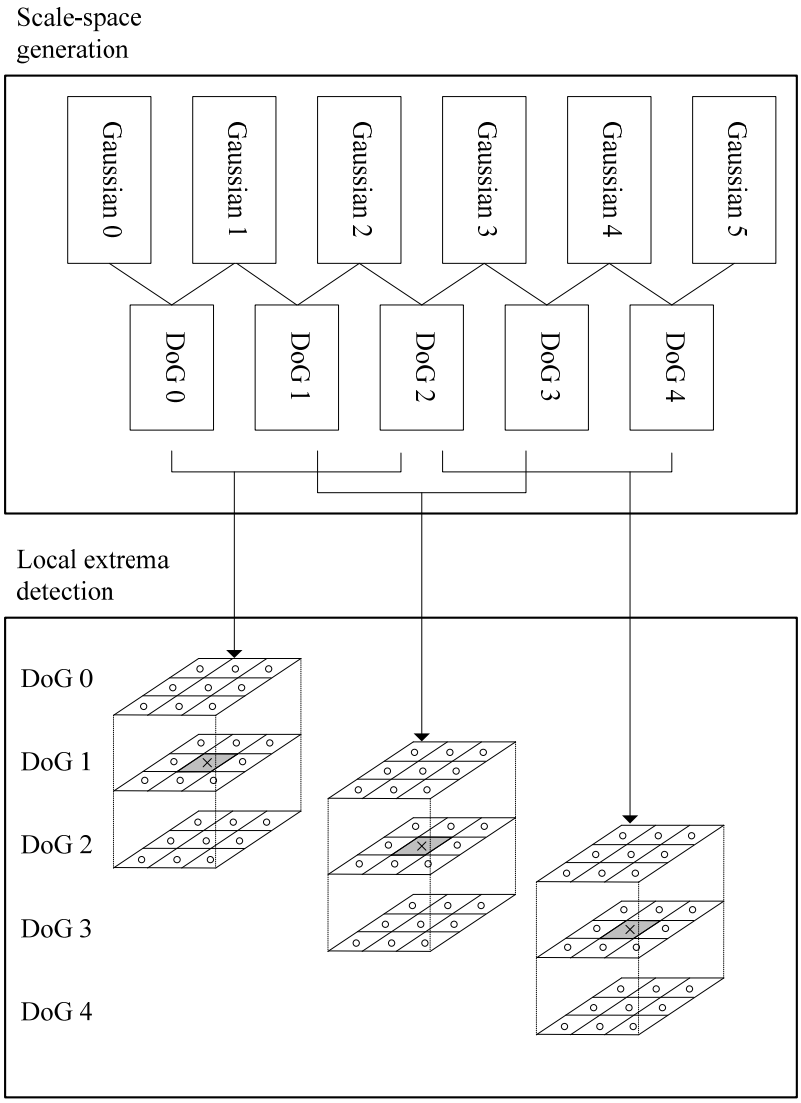


그림 2.4 Local extrema detection

2.1.3 Keypoint Localization

Keypoint Localization 단계에서는 keypoint 의 위치를 세부적으로 조정하거나, low contrast 값을 가지는 경우 또는 edge 주변에 위치한 keypoint 를 제거하는 과정을 수행한다. 이전 단계에서의 local extrema 을 sub-pixel, sub-scale 단위까지 세부적으로 찾기 위해서, 주변 DoG 값을 이용한 interpolation 을 수행한다. 이것은 식 (2.4)와 같이 DoG 함수의 Taylor 시리즈를 이차항까지 전개한 수식을 사용한다. 여기서 $\mathrm{D(x)}$ 는 해당 keypoint 의 후보점을 원점으로 하여 계산한다. 또한, 벡터 $\mathrm{x} = [x, y, \sigma]^T$ 는 후보점에서의 sub-pixel 단위의 offset 을 의미한다.

$$D(\mathrm{x}) = D + \frac{\partial D^T}{\partial \mathrm{x}}\mathrm{x} + \frac{1}{2}\mathrm{x}^T\frac{\partial^2 D}{\partial \mathrm{x}^2}\mathrm{x} \tag{2.4}$$

Extrema 의 위치 $\hat{\mathrm{x}}$ 은 식을 x 에 대해서 미분한 후 0 으로 두면, 식 (2.5) 와 같은 결과를 생성하는데, 이때의 x 를 $\hat{\mathrm{x}}$ 로 한다.

$$\hat{\mathrm{x}} = -\frac{\partial^2 D}{\partial \mathrm{x}^2}^{-1}\frac{\partial D}{\partial \mathrm{x}} \tag{2.5}$$

이 값이 식 (2.4) 로 표현된 interpolation 식이 극값이 되는 sub-pixel offset 이다. 만약 식 (2.5) 의 값이 벡터 x 의 어떤 방향에서라도 크기가 0.5 을 넘는 것이 있으면, keypoint 후보점의 위치를 조정하게 된다.

또한 식 (2.6) 으로 표현된 extrema 위치에서 함수값이 low contrast 를 가지는 불안정한 위치의 extrema 값은 제거하게 된다. 논문[9] 에서는

$|D(\hat{x})| < 0.03$ 의 값을 가지는 keypoint 의 위치를 keypoint 후보에서 제외한다.

$$D(\hat{x}) = D + \frac{1}{2}\frac{\partial D^T}{\partial x}\hat{x} \tag{2.6}$$

여기서 사용되는 contrast threshold 값은 최종적으로 선택되는 keypoint 의 수에 관련이 있는 변수이다. 또한 이와 같은 threshold 값은 하나의 이미지에 대해서, 동일하게 적용되는 값이다. 따라서, contrast 가 강한 영역 주변의 keypoint 들은 많이 유지되고, contrast 가 약한 영역 주변의 keypoint 는 제거된다.

2.1.4 Orientation assignment

Orientation 의 변화에 대해 영향을 받지 않는 descriptor 를 생성하기 위해서, 해당 keypoint 에서의 dominant 한 orientation 을 계산하게 된다. 각 keypoint 에 해당하는 scale 의 Gaussian filtering 된 이미지 $L(x, y, \sigma)$ 을 사용하여, keypoint 주변의 gradient magnitude $m(x,y)$와 orientation $\theta(x,y)$ 을 계산한다. Keypoint 위치에서의 gradient 는 식 (2.7) 식 (2.8)과 같이 주변 픽셀 값의 뺄셈으로 계산된다.

$$\Delta_x = \frac{1}{2}\big(L_i(x+1,y) - L_i(x-1,y)\big) \tag{2.7}$$

$$\Delta_y = \frac{1}{2}\big(L_i(x,y+1) - L_i(x,y-1)\big) \tag{2.8}$$

$$m(x,y) = \sqrt{\Delta_x^2 + \Delta_y^2} \qquad \theta(x,y) = tan^{-1}\left(\frac{\Delta_y}{\Delta_x}\right) \tag{2.9}$$

2.1.5 Descriptor generation

그림 2.5 는 keypoint descriptor 를 계산하는 과정을 보여 준다. 이전 절에서 계산한 orientation 을 통해 local patch 의 dominant orientation 을 계산한다. 이를 위해 keypoint 주변 local patch 에서 orientation histogram 을 사용하는데, orientation histogram 은 local patch 내의 Gaussian filtering 된 이미지 $L(x, y, \sigma)$ 의 gradient orientation 으로 생성된다. Orientation histogram 은 360 도를 표현하기 위해서 36 개의 bin 으로 나뉘어 있다. Histogram 을 생성하기 위해서 keypoint 를 중심으로 keypoint 의 scale 의 1.5σ 에 해당하는 영역 내의 값을 사용하게 된다. 영역 내의 gradient 값은 해당 위치에서의 scale 의 1.5σ 에 해당하는 Gaussian-weighted circular window 에 의해 가중치가 가해진 후 그 값이 orientation 이 속하는 bin 에 축적된다.

Orientation histogram 에서의 peak 값은 local gradient 의 dominant direction 을 나타낸다. 선택적으로 local peak 값의 80% 이상인 direction 에 대해서, 추가적인 dominant orientation 으로 설정하기도 한다. 따라서, 이러한 경우엔 동일한 위치, scale 을 가지고, 다른 orientation 을 가지는 복수 개의 keypoint 가 가능하게 된다.

이후, dominant orientation 으로 선택된 가장 큰 각도를 기준으로 정규화하여, 회전에 무관한 orientation 으로 변환된다. 그것을 다시 8 개의 각도로 통합하고, 그것을 히스토그램으로 만들어 한 영역의 descriptor 로 사용한다

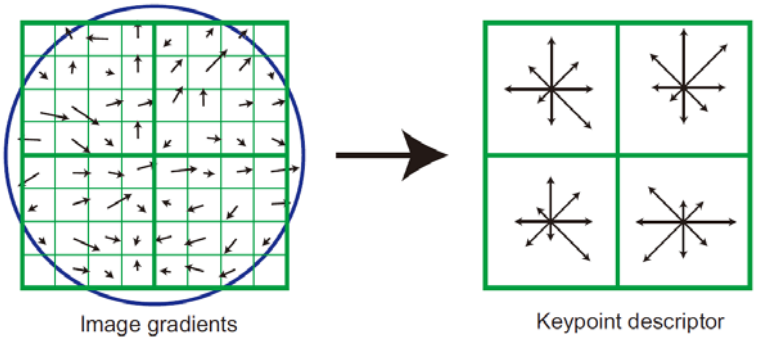


그림 2.5 Keypoint descriptor 생성 방법

그림 2.5 는 keypoint 주변의 descriptor 를 생성하는 예를 보여준다[9]. Gradient 이미지는 8x8 sample 들로 구성되어 있고, 이를 통해 2x2 descriptor array 를 생성하는 과정을 보여준다. 본 연구에서는 16x16 array 를 사용해서, 4x4 descriptor 를 생성하였다. 각 descriptor array 는 8 개의 bin 을 가진다. 따라서 실험에서는 4x4x8 = 128 개의 feature vector 를 가지는 descriptor 를 생성하게 된다.

2.1.6 Descriptor correspondence

본 단계에서는 이전 절에서 생성된 descriptor 들을 통해 이미지간 matching 을 수행한다. 본 연구에서는 두 개의 이미지에서 descriptor 간의 유사성을 통해서, 가장 유사한 descriptor 로 찾은 descriptor 쌍을 correspondence 라고 하였다. 또한, correspondence 내에는 많은 수의 incorrect correspondence 를 포함하기 때문에, correspondence 로부터 correct correspondence 를 찾는 과정을 matching 으로 사용하였다.

각 keypoint 의 descriptor 에 대해 correspondence 를 찾는것은 다른 이

미지의 keypoint database 중 가장 유사한 descriptor 를 찾는 것이다. 두 개의 descriptor vector $\mathbf{u}, \mathbf{v}$ 간에 유사성은 Euclidean distance 로 표현하고, 식 (2.10)과 같이 정의된다.

$$d(\boldsymbol{u}, \boldsymbol{v}) = \left(\sum_i (u_i - v_i)^2 \right)^{1/2} \tag{2.10}$$

SIFT 의 descriptor 는 128 descriptor vector 를 갖기 때문에, 128 차원의 Euclidean distance 로 계산된다[9]. 이와 같이 계산된 Euclidean distance 를 사용하여 correspondence 를 구성하는데는 몇 가지 방법들이 있다. 먼저, NN(Nearest Neighbor) 방법은 최소의 Euclidean distance 를 갖는 descriptor 를 선택하는 방법이다. 또한 NNDR(Nearest Neighbor Distance Ratio)을 사용하는 방법은 식 (2.11) 과 같이 nearest neighbor 를 가지는 descriptor 의 Euclidean distance (d_1) 과 second nearest neighbor 를 가지는 descriptor 의 Euclidean distance(d_2) 간의 distance ratio 를 사용하는 방법이다.

$$NNDR = \frac{d_1}{d_2} \tag{2.11}$$

따라서 NNDR 값은 1 보다 작은 값을 가지게 되고, NNDR 의 값이 작을수록, 즉 두 descriptor 간의 차이가 분명히 발생하는 경우에 좋은 correspondence 로 판단한다.

2.2 FAST (Features from Accelerated Segment Test)

FAST (Features from Accelerated Segment Test)[22]은 이미지내에 존재하는 corner 를 keypoint 로 찾는 detector 방법이다. 특히 FAST 는 빠른 속도로 코너 검출이 가능하다. FAST 에 의한 코너 검출 방법은 다음과 같은 단계를 거쳐서 생성된다.

먼저 keypoint 를 찾기 위해 전체 이미지에서 corner 로 판단할 후보 위치 p 의 주변을 원 형태로 16개 픽셀을 조사한다. Corner 후보 p 에 대해서 주변 조사를 위한 픽셀 위치는 그림 2.6 과 같이 구성된다[22]. 16개의 픽셀을 각각 후보 p의 밝기 정도 I_p와 미리 설정한 threshold 값 t를 더한 것 보다 밝으면 그 픽셀은 brighter로 두고, 값 t를 뺀 것 보다 어두우면 그 픽셀은 darker로 둔다. 그 외의 픽셀은 similar로 둔다. 이를 수식으로 나타내면 식 (2.12)과 같다. 원에 해당하는 픽셀의 위치에 따라 값 x를 가지고 밝기는 $I_{p\to x}$ 로 둔다.

$$S_{p\to x} = \begin{cases} d, & I_{p\to x} \le I_p - t \\ s, & I_p - t \le I_{p\to x} \le I_p + t \\ b, & I_p + t \le I_{p\to x} \end{cases} \tag{2.12}$$

후보 p의 주변 16개의 픽셀중 연속된 n개 이상의 픽셀이 darker이거나 brighter이면 후보 p를 corner로 설정한다. Corner가 되기 위한 최소 n의 값은 9이고, 충분한 corner를 검출하기 위한 최대 n의 값은 12이다. 이를 반복하여 전체 이미지에서 corner를 검출해 낸다.

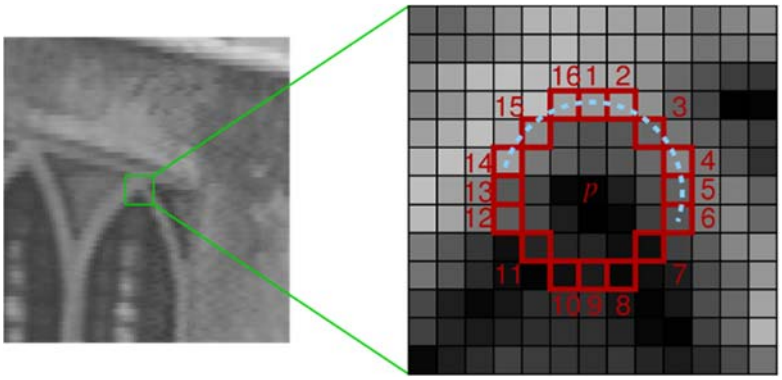


그림 2.6 FAST corner detector

검출해 낸 corner 들을 최종 keypoint 로 선정하기 전에, 위와 같은 조건을 만족하는 경우의 corner 들을 추출하면 한 개의 corner에 keypoint 가 여러 개 검출된다. 따라서 비슷한 영역의 존재하는 keypoint 의 개수를 줄이는 알고리즘이 필요한데, 이 알고리즘을 non-maximal suppression 이라고 한다. Non-maximal suppression 알고리즘은 후보 p마다 score를 매긴 후 1 픽셀위치 만큼 차이가 나는 두 후보 p의 score를 비교하여 score가 작은 후보 p는 corner에서 제외하는 방법을 사용한다. Score 는 식 (2.13) 와 같이 계산된다.

$$score = max \begin{cases} \sum \left(p \text{ 의주변값} - p\right) & if \left(p \text{ 의주변값} - p\right) > t \\ \sum \left(p - p \text{ 의주변값}\right) & if \left(p - p \text{ 의주변값}\right) > t \end{cases} \quad (2.13)$$

위와 같이 FAST detector 를 통해서, 추출한 keypoint 들은 영역 내에 발생한 keypoint 의 수를 합해서, 영역 내에 이미지의 특성을 예측하기 위한 방법으로 사용된다.

2.3 이전 연구

일반적인 recognition 시스템은 1 장에서 설명한 바와 같이 크게 keypoint 를 찾고, 이를 기술하는 descriptor 를 만드는 과정(feature extraction)과 이를 통해 matching(feature matching) 하는 단계로 구성되어 있다. 이 절에서는 각각의 단계에서 효과적인 연산을 수행하기 위해서, 기존에 연구되었던 내용들을 소개하고, 기존 방법의 문제점을 살펴본다.

2.3.1 Keypoint selection and spatial distribution

Keypoint detection 단계에서 생성된 많은 수의 keypoint 는 이후 단계의 연산 과정인 descriptor 생성 단계와 matching 단계에서 keypoint 수에 비례한 연산과정을 필요로 한다. 이미지에서 생성되는 많은 수의 keypoint 는 이미지의 특징을 많이 추출한다는 측면에서는 좋지만, 이후 단계의 연산에 부담이 될 뿐만 아니라, 동시에 정확한 위치를 찾아야 하는 응용에서는 비슷한 위치에 발생하는 여러 개의 keypoint 로 인해 잘못 correspondence 가 생성될 가능성이 높아진다. 또한, 영역을 찾는 응용의 경우에는 과도하게 많은 수의 keypoint 가 특정 영역에 존재하기 보다는 keypoint 의 분포가 고르게 되어 있는 경우가 더 유리하다. 이와 같은 필요성에 의해서, keypoint 수를 조절하기 위한 연구들이 진행되어 왔다[27].

먼저 단순히 발생된 keypoint 중 N 개의 가장 강한 특성을 가지는 feature point 를 찾는 방법이 있다[23]. 이와 같은 방법은 가장 특성이 강한 영역 주변의 keypoint 만 선택되게 된다. 따라서, 특정 영역에만 이미지를 표현하는 keypoint 가 남게 되기 때문에, 상대적으로 특성은 낮지만, 필요한

위치를 기술할 keypoint 가 없어지는 문제가 발생한다. 따라서 특히 image mosaicking, robot navigation, scene recognition 응용에서와 같이 keypoint 간 상대적 위치를 고려하는 응용에서는 특정영역에만 keypoint 가 분포되어 있는 경우엔 불리하게 된다[23][24]. 이와 같은 경우에서도 볼 수 있듯이 단순히 keypoint 의 수뿐만 아니라, keypoint 의 분포 또한 고려가 되어야 한다. 이를 개선하기 위해서, 전체 이미지에 일정하게 분포된 keypoint 를 선택하기 위해서, ANMS(Adaptive Non-Maximal Suppression) 방법을[25] 사용하여, 발생된 keypoint 에 대해서, 일정 범위 내에서 가장 강한 특성을 가지는 keypoint 를 선택하는 방법이 있다. 또, 발생된 keypoints 중, 사전에 계산된 영역에서 하나씩만 선택하는 방법[26][29], 발생된 keypoint 를 트리 형태의 구조를 이용해서 선택하는 방법[28]등을 사용해서 전체 이미지 내에서 발생하는 keypoint 를 줄이는 방법에 대한 연구들이 진행되었다. 이와 같은 방법들은 keypoint 의 수를 조절하여, 이후 단계에 수행되는 descriptor 생성 단계나 matching 단계의 수행 시간을 줄이는데 효과적으로 사용 가능할 뿐만 아니라, keypoint 들이 전체적으로 고르게 분포되어서 전체 이미지를 효과적으로 표현할 수 있다.

그러나 이와 같은 방법은 발생된 keypoint 결과에 의존성을 가지고 있다. 즉, keypoint 가 생성된 이후 발생된 keypoint 의 분포에 대해서 적용이 가능한 방법들이다. 따라서, 하드웨어로 구현할 경우 이러한 방법은 병렬적으로 처리가 불가능하기 때문에, 기존에 병렬화된 pipeline 구조를 유지할 수 없게 된다. 본 연구에서는 하드웨어에 적용 가능한 keypoint selection 에 대한 연구를 제안하였다.

2.3.2 Clustering 기반의 feature matching

SIFT 에서 생성한 local patch descriptor 의 유사성만을 이용하여, correspondence 를 찾으면, 부분적으로 유사한 다른 부분의 descriptor 로 correspondence 가 이루어질 수 있다. 따라서, 초기의 correspondence 로부터 inlier 와 outlier 를 구분하는 방법들에 대한 연구들이 진행되고 있다.

이전 논문[9] 에서는 이미지를 rotation, translation, scale 의 변화만을 보이는 rigid scene 으로 가정하고, RANSAC(RANdom Sample Consensus) 방법을[9][11] 사용하여, 한 이미지에 point 들이 다른 이미지에서 어떤 위치에 대응하는지에 대한 affine transform 을 추정하여, affine transform 에 부합하는 correspondence 를 inlier 로 선택하고, 나머지를 outlier 로 선택하는 방법을 사용한다. 그러나 이와 같은 방법은 non-rigid image 의 변형이나, affine transform 에 의해서 표현되지 않는 complicated scene 같은 경우에는 효과적이지 못하게 된다. 따라서 최근 들어 non-rigid image 변화에 적합한 matching 알고리즘들이 연구되고 있다[31][32][33].

Non-rigid 한 이미지에서의 matching 을 위해 feature 간 geometric 한 정보를 사용하는 방법들이 있다. 이는 이미지 내의 keypoint 간 distance 나 angle 이 상대적으로 불변함을 이용하여, keypoint 들로 구성된 graph 를 사용하여 pairwise 한 대응 쌍을 찾거나, triples of points set 을 찾는 방법을 사용한다. 이러한 복수개의 keypoint 들을 사용해서 수행되는 high-order matching 은 local feature 의 유사성뿐만 아니라 keypoint 간의 위치 관계를 사용하여 matching 을 수행한다. 그러나 이러한 방법은 많은 수의 correspondence 가 존재하는 경우, outlier 의 조합에 의해서도 distance 나

angle 간의 불변한 성질을 만족하기 때문에, 잘못된 결과를 생성할 가능성이 많게 된다. 또한 이러한 방법은 연산의 복잡도가 높은 문제가 있다. 이러한 문제점을 개선하기 위해서 최근에 feature 간의 clustering 을 이용하여 신뢰성 높은 feature set 을 얻는 방법들이 있었다. 이와 같은 clustering 방법들은 비교적 정확한 결과를 제공하지만, 이미지 내의 모든 feature correspondence 를 대상으로 반복적으로 clustering 과정을 수행하기 때문에, correspondence 의 수가 늘어나면 기하급수적으로 수행 시간이 증가하게 된다[35][38].

본 연구에서는 이와 같이 clustering 에 의한 feature matching 방법에서 발생하는 연산의 복잡도를 개선하기 위해 region-constrained clustering 방법을 제안한다.

2.3.3 Watershed Transform

많은 응용 분야에서 실시간 이미지 segmentation 을 필요로 하고 있으며, 여러 가지 이미지 segmentation 알고리즘 가운데, watershed 알고리즘[39]은 불분명한 경계를 가지는 object 를 효과적으로 구분할 수 있는 방법이다[40]. 그러나 watershed 알고리즘은 전체 이미지에 대하여 gradient 를 기준으로 sorting 을 하고 이 순서에 따라 픽셀별로 주변값과의 관계를 조사하여 순차적으로 연산을 하기 때문에, 연산량이 많고 병렬화가 어렵다. 따라서, 입력 이미지의 크기가 커짐에 따라 watershed 알고리즘의 수행 속도를 급격히 저하시키게 되고, 그에 따라 실시간 처리가 어렵게 된다[43][44]. 이와 같은 segmentation 은 개별 결과뿐만 아니라, 시스템에서 전처리 과정이나

정보를 제공하는 단계로도 많이 사용되기 때문에, 빠른 연산이 필요하다.

이전 연구에서는 이미지의 변화된 부분만을 재계산하여, segment 이미지를 생성하는 방법이 있다. 이러한 방법은 이전에 계산된 부분과, 새로 계산해야 하는 부분을 설정하는데, 새로 계산하는 부분이 블록의 집합군의 형태로 나타나게 된다. 이 방법은 새로 계산하는 블록과 기존 계산된 블록과의 경계면에서 전체 이미지의 segmentation 과 다른 over-segment 된 결과를 생성한다. 또한 블록의 형태가 임의의 크기로 구성된 블록의 집합군의 형태를 가지므로 하드웨어 구현에 적합하지 않다[42]. 본 연구에서는 독립적으로 동일한 형태의 블록 단위로 수행이 가능한 블록 기반의 watershed 방법을 제안하고, 블록 경계에서 발생하는 부정확한 결과를 향상시키는 방법을 제안하였다.

제3장 Adaptive keypoint generation 하드웨어 구현

이번 장에서는 adaptive keypoint generation 을 위한 SIFT 하드웨어 구조에 대해 제안한다. 이를 위해서, 하드웨어 구조에 적합한 keypoint 조절 방법에 대해서 살펴보고, 제안된 하드웨어 구조에 대해서 설명한다.

3.1 SIFT 하드웨어 속도 향상을 위한 요소

SIFT 는 다른 여러 가지 local feature 방법 중 다양한 이미지 변형에 강인한 특성을 보이기 때문에 가장 널리 사용되고 있다[13][15][20]. 하지만 SIFT 는 많은 연산을 요구할 뿐만 아니라, 많은 양의 메모리를 필요로 하기 때문에, 실시간으로 구현하는데 어려움이 있다. 따라서 SIFT 의 연산을 간략화 해서, 연산 속도를 높이기 위한 연구들이 있어 왔다[14][19]. 또한 FPGA 같은 하드웨어 가속기를 사용하여 연산 속도를 높이는 연구들도 이루어져 왔다[16][17][18]. 이러한 하드웨어를 사용한 접근 방법들은 연산의 병렬처리나 메모리 재사용 방법 등을 통해 속도와 효율성을 높이는 방향으로 연구가 진행되어 왔다. 그럼에도 불구하여, 여전히 실시간 구현에 어려움을 가지고 있다. 본 연구에서는 하드웨어로 구현된 SIFT 에 적합한 속도 향상 방법을 제안한다.

이를 위해 먼저 SIFT 연산에 대한 블록별 특성을 살펴보면 그림 3.1 과 같이 연산량에 영향을 미치는 요소들이 서로 다름을 알 수 있다. Keypoint

detection 은 전체 연산량이 입력 이미지의 크기와 관련이 있는 부분이다. 따라서 입력 이미지의 크기만 관련이 있고, 이미지내에 존재하는 특성과는 관련이 없는 부분이다. 반면 descriptor generation 부분과 feature matching 부분은 전체 연산량이 keypoint detector 에서 발생한 keypoint 의 수와 관련된 부분으로 입력 이미지의 크기와는 관련이 없는 부분이다.

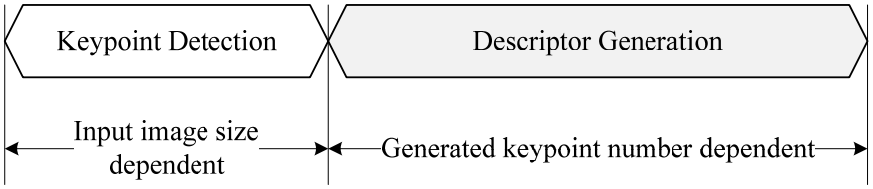


그림 3.1 블록별 연산량에 영향을 미치는 요인

하드웨어 구현된 SIFT 알고리즘에서는 keypoint detection 이 다 끝나기 전에 생성된 keypoint 를 사용하여, descriptor generation 부분을 시작할 수 있다. 따라서, 그림 3.2 와 같이 병렬 처리하도록 구성되어 있다. 그럼에도 descriptor 생성 부분은, keypoint detection 부분에 비해 많은 양의 연산을 필요로 하므로, keypoint detection 이 완료된 이후에도 descriptor generation 모듈을 수행하기 위한 상당한 시간이 필요하게 된다.

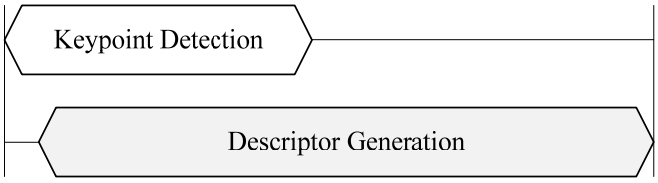


그림 3.2 병렬화된 keypoint detection 과 descriptor generation

앞에서 살펴본 바와 같이 keypoint 의 수는 전체적인 속도에 영향을 미치는 요소이기 때문에 keypoint detection 단계에서 keypoint 수를 줄임으로써, 전체적인 속도 향상을 하는 연구들이 있어 왔다. 그러나 단순히 전체 개수만 줄이는 방법을 적용할 경우에는 이미지내에 특징을 기술할 keypoint 가 특정 영역에만 존재하기 때문에, 이미지의 특성을 고려하여, keypoint 들이 고르게 분포할 필요가 있다.

이전 연구 방법들은 keypoint 수에 의해 발생하는 연산의 복잡도 문제를 해결하기 위해서, keypoint 의 생성이 완료된 이후에, 생성된 keypoint 들을 sampling 하는 방법을 사용하였다. 따라서, 그림 3.3 와 같이 구성된 병렬화된 SIFT 하드웨어 구조에 적용하기 어려운 문제가 있다.

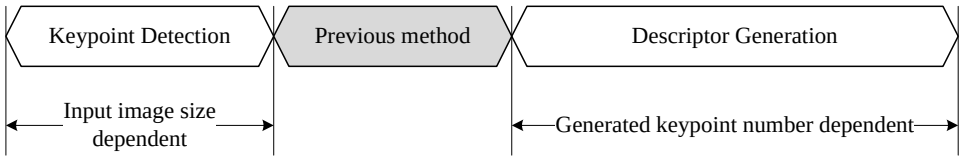


그림 3.3 이전 연구 방법

이와 같은 문제점 때문에, 하드웨어에 적합한 keypoint 를 조절할 수 있는 방법이 필요하다.

3.2 Hardware 에 적합한 keypoint 조절 방법

이전의 연구 방법들이 keypoint 생성 이후에 keypoint 조절하기 때문에, 병렬화된 하드웨어에 적용하기 어려운 문제점이 있었다. 따라서, 하드웨어에 적용하기 위해서는 keypoint 생성 단계에서 keypoint 수를 조절하는 방법이 필요하다. SIFT 는 DoG 이미지의 local extrema 로부터 keypoint localization 단계를 거쳐서 최종 keypoint 를 생성하게 된다. Localization 단계에서는 keypoint 주변이 low contrast 를 가지는 keypoint 를 제거하게 되는데, 이 과정은 최종 keypoint 수와 관련이 있게 된다. 이전 논문에서는 [9] $|D(\hat{x})| < 0.03$ 조건을 전체 이미지에 적용하여, 최종 keypoint 로 선택하였다.

Contrast threshold 값에 따른 keypoint 발생 수를 살펴보면, threshold 변화 비율에 비례하여, keypoint 수가 달라지는 것을 알 수 있다. SIFT 의 경우엔 각각 octave 별로 keypoint 가 생성되는데, octave 별로도 같은 경향성을 보임을 알 수 있다.

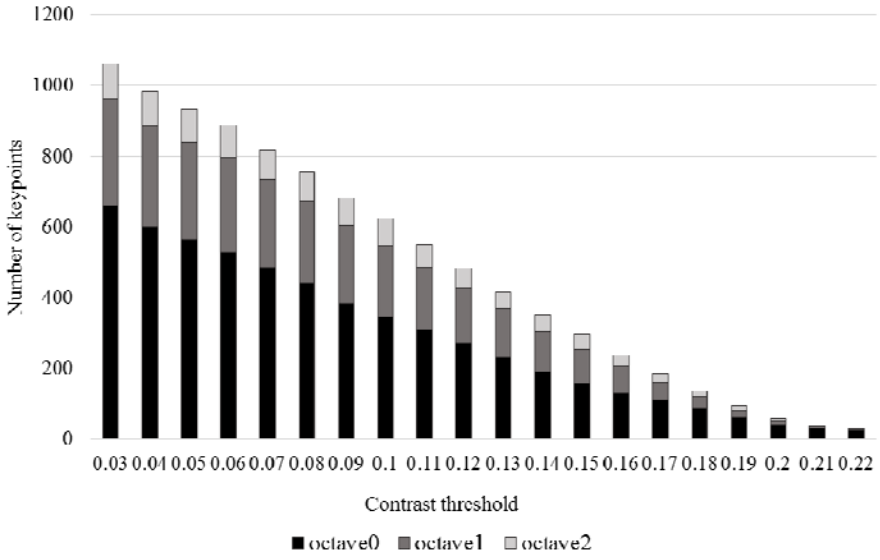


그림 3.4 Contrast threshold 에 따른 keypoint 수 변화

이와 같이 contrast threshold 는 keypoint 를 수를 조절할 수 있는 요소로 작용하고, keypoint detection 동작에 적용이 되어, 하드웨어로 구현된 SIFT 에서 keypoint 조절을 위한 방법으로 사용 가능 하다. 그러나 contrast threshold 를 통해 keypoint 의 개수는 조절이 가능하지만, 이전 연구의 문제점에서도 살펴본 바와 같이 keypoint 분포를 고려하면, keypoint 의 수를 줄이기 위해서, 이미지에 동일한 contrast threshold 값을 적용하는 것은, contrast 가 강한 영역의 주변의 keypoint 만 남게 되고, low contrast 영역 주변의 keypoint 는 없어지게 된다. 따라서, 이미지를 표현할 feature 들이 일부의 영역에만 존재하게 된다. 따라서 matching이 될 가능성이 있는 전체 local patch 가 줄어들게 된다. 이러한 문제점 때문에, keypoint 의 수를 줄이는 문제에서는, keypoint 의 분포도 함께 고려가 되어야 한다.

이와 같이 상대적인 위치 분포를 유지하면서, keypoint 를 줄이기 위해서는 특정 영역에 많은 수의 keypoint 가 발생하는 영역에 대해서만 keypoint 를 조절하는 방법이 필요하다. 그러나 SIFT 의 keypoint 를 생성하기 전에는 특정영역에 keypoint 가 많이 발생하는지 여부를 알 수 없기 때문에 어떤 영역의 keypoint 를 줄여야 할지에 대한 정보도 알 수 없게 된다. 따라서 SIFT keypoint 생성 이전에 상대적으로 SIFT keypoint 가 많이 발생할 부분을 예측하면, 영역별로 다른 contrast threshold 값을 통해서 low contrast 주변의 keypoint 는 유지하면서, high contrast 주변에 많이 발생하는 keypoint 만 조절을 할 수 있게 된다.

3.3 FAST detector 를 통한 이미지 특성 예측

SIFT 는 DoG(Difference Of Gaussian)이미지를 사용하여, DoG 의 local maxima 위치에서 keypoint 를 생성한다. 이러한 DoG 는 edge 나 blob 주변에서 강하게 반응하기 때문에, edge나 blob 주변에서 SIFT의 keypoint 들이 많이 나타나게 된다. SIFT 에서는 이와 같은 keypoint 를 생성하기 위해서, scale 단계에 따라서, 여러 번의 Gaussian filtering 과정을 수행하게 된다.

다른 feature detector 들은 다른 과정을 통해 이미지에서 특징점들을 생성한다. 이와 같은 feature detector 들이 생성하는 keypoint 의 위치들은 서로 일치하지 않지만, 발생된 keypoint 의 분포들에는 서로 상관성을 가지고 있다.

예를 들어 단일 색상의 배경 영역 부분에서는 keypoint 의 발생 분포가 낮고, 건물과 같은 많은 수의 edge 와 corner 가 존재하는 영역에서는 keypoint 의 발생 분포가 높게 생성된다. 이는 feature 가 물체의 distinctive 한 특징을 나타내기 때문에, 주변에 이미지 변화가 없는 영역보다는 이미지 변화가 많은 영역에 많이 나타나게 된다. 따라서 일정 영역 기준으로 보면, 다른 종류의 feature detector 에서 생성된 keypoint 간에는 keypoint 의 위치를 다르지만, 영역내의 발생 분포 밀도 측면에서는 상관성을 가지고 있다. 따라서 SIFT 에서 발생하는 keypoint 발생 밀도를 예측하는데, 다른 종류의 detector 를 결과를 활용할 수 있다. 특히 corner 는 서로 다른 방향성을 가진 edge 의 교차점에서 관찰되는데, 일정 범위 내에서 살펴보면, 결국 여러 개의 edge 로 구성된 영역 주변에서 SIFT 의 keypoint 가 많이 발생하게 된다.

본 연구에서는 SIFT 의 keypoint 의 발생 분포를 예측하기 위해서, corner detector 의 한 방법인 FAST(Features from Accelerated Segment Test)[22] 를 사용하였다. 또한 FAST detector 는 빠르게 keypoint 를 생성할 수 있기 때문에, SIFT 의 keypoint 의 분포를 사전에 예측하는 방법으로 효과적으로 사용 가능하다.

3.3.1 SIFT keypoint 와 FAST keypoint 분포의 상관성

본 절에서는 블록 단위로 발생하는 SIFT keypoint 와 FAST keypoint 분포 간의 상관성을 설명한다. FAST detector 는 현재 픽셀 위치에서 3 픽셀 떨어진 위치의 픽셀들을 원 형태로 조사한다. 반면 SIFT 의 경우 edge 주변이나 blob 의 center 위치에서 keypoint 가 생성되게 된다. 이때, 발생 위치는 Gaussian kernel size 에 따라서 달라지게 된다. 따라서 두 detector 간에 발생하는 위치는 서로 다르게 된다. 그러나 블록별 분포간에는 연관성을 가지고 있다.

먼저 두 종류의 특성이 다른 이미지들에 대해서 SIFT 와 FAST 에 의해서 발생하는 keypoint 에 대한 분포를 도출하고, 분석을 진행하였다. 그림 3.5 의 자연 영상 이미지와 그림 3.6 의 인공 구조물에 대한 이미지들을 가지고 FAST 와 SIFT 에서 생성된 keypoint 들간에 블록단위 분포의 상관성을 실험하였다. 각각 종류의 이미지들은 500 개의 dataset 으로 구성하였다.

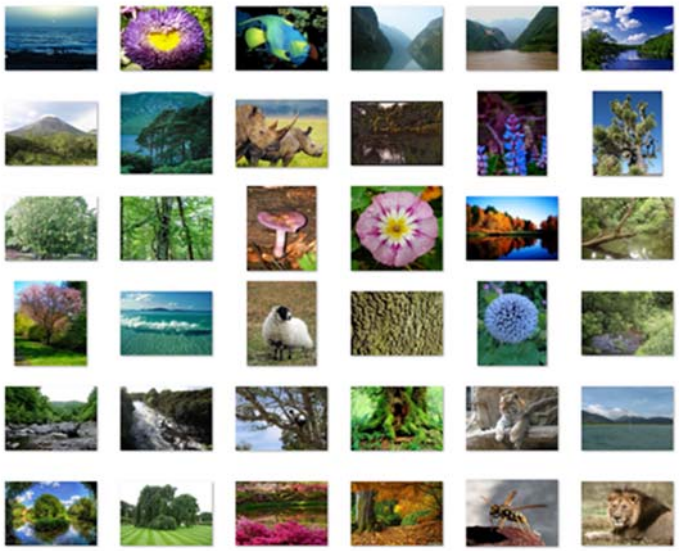


그림 3.5 자연영상 이미지 예

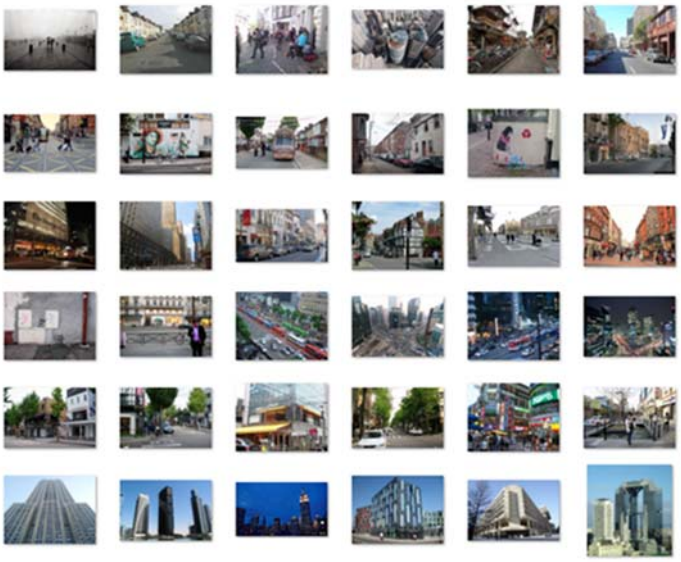


그림 3.6 인공 구조물 이미지 예

표 3.1 FAST 와 SIFT 에 의한 블록별 keypoint 분포의 상관성

		FAST 에 의한 블록내 이미지의 복잡도	
		높다	낮다
SIFT 에 의한 블록내 이미지의 복잡도	높다	I	II
	낮다	III	IV

블록별로 FAST keypoint 분포에 따라서 SIFT 의 keypoint를 예측하는 경우엔 표 3.1 과 같이 4가지의 경우가 생기게 된다. Case I 은 FAST keypoint 의 갯수에 의해서 블록내 이미지가 복잡하다고 판단된 영역에 실제로 SIFT 에 의해서도 keypoint 가 많이 나오는 경우로 SIFT 에 의해서도 복잡도가 높게 판단된 경우이다. Case II 은 FAST 에 의해서 블록내 이미지의 복잡도는 낮게 판단되었으나, SIFT 에 의해서는 복잡도가 높게 나오는 경우이다. Case III 는 FAST 에 의해서는 복잡도가 높은 영역이 SIFT 에 의해서는 복잡도가 낮게 판단된 경우이며, Case IV 는 FAST 와 SIFT 에 의해 모두 블록내 이미지의 복잡도가 낮은 영역으로 판단된 경우이다.

따라서, Case I, IV 는 맞게 판단된 블록으로 SIFT 에 의해서 발생하는 keypoint 가 맞게 조절되는 경우이다. 반면, Case II 는 FAST 에 의해서 복잡도가 낮은 영역으로 판단이 되었지만, 실제로는 SIFT 에 의해서는 keypoint 가 많이 발생하는 영역으로 keypoint 가 개수가 줄어야 하지만, 줄어들지 못하는 영역이다. 반대로 Case III 는 FAST 에 의해서 복잡도가 높은 영역으로 판단하고, SIFT 의 keypoint 의 수를 줄이지만, 실제로는 SIFT 의 keypoint 발생 분포가 낮은 영역으로 영역을 기술할 keypoint 가 사라질

수 있는 영역이다.

실제 정확도는 전체 블록 중 Case I, IV 로 판단하는 블록은 맞게 판단한 부분으로 하고, Case III 는 과도하게 keypoint 줄이는 영역, Case II 는 keypoint 를 줄일 필요가 있지만, 줄이지 못하는 영역으로, Case II, III 가 상관성을 정확하게 예측하지 못한 부분이 된다.

예를 들어 그림 3.7 (a) 의 원본 이미지에 대해서 (b)는 FAST 의 keypoint 수에 따라서 블록의 복잡도를 분류한 결과이고, (c) 는 SIFT 에 의해서 블록의 복잡도를 분류한 결과이다. 좌상단이 복잡도가 낮은 블록들이고, 우하단으로 갈수록 이미지의 복잡도가 높은 블록이다. 그림 3.7(b), (c) 는 대체적으로 비슷한 경향성을 보이지만, 블록간 정확히 분류가 일치하지 않게 된다.

표 3.2 는 실험 영상에서 이러한 차이를 설명한다. 실험은 FAST 에서 블록에서 발생하는 평균적인 keypoint 의 수보다 30% 많이 발생하는 블록을 복잡도가 높은 블록으로 선정하였다. 두 종류의 테스트 이미지들에 대해서, 맞게 예측한 경우(Case I, IV) 는 자연영상 이미지의 경우 77%, 인공 구조물 이미지의 경우 79% 정도로 비슷한 상관관계 예측결과 수치를 보여준다.

표 3.2 FAST 와 SIFT 에 의한 블록별 상관관계 예측 결과

	Case I, IV	Case II	Case III
자연영상	77%	11.5%	11.5%
인공구조물영상	79.1%	10.5%	10.5%

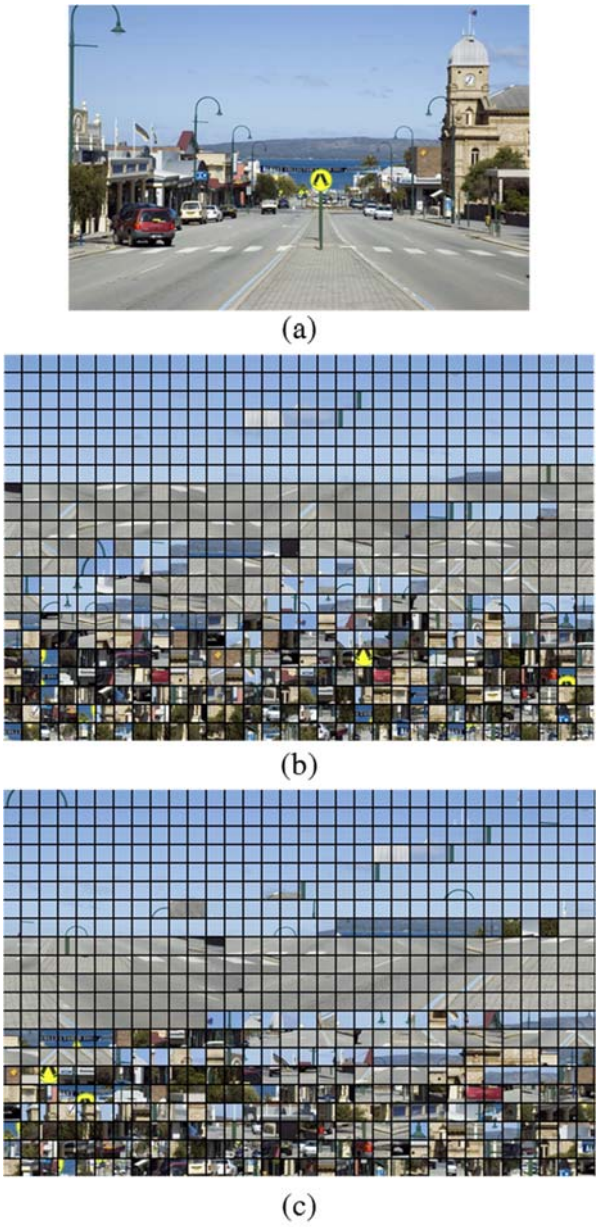


그림 3.7 블록별 복잡도에 따른 분류 (a) 원본 영상 (b) FAST 에 의한 분류 (c) SIFT 에 의한 분류

3.4 Adaptive keypoint generation 하드웨어 구조

전체 SIFT 하드웨어는 그림 3.8 와 같이 크게 4 부분의 기능별 모듈로 구성되어 있다. Gaussian filter bank 에서는 외부 메모리에 저장된 영상 데이터를 Gaussian filter bank 내부 source line buffer 에 저장하고, 저장된 이미지는 모든 Gaussian filter 에 동시에 전달되어 다양한 Gaussian filter kernel 에 대해서 filtering 이 동시에 이루어지게 된다. Filtering 된 이미지로부터 생성된 DoG 이미지는 keypoint detector 로 전달되며, 또한 생성된 scale-space 정보는 descriptor 를 생성할 때 이용하기 위해서 memory 에 저장된다. Keypoint detector 모듈에서는 DoG 이미지로부터 local extrema 를 찾고, contrast check 단계와 edge 근처에 있는 값을 제거하여, 최종적인 keypoint 값을 생성하게 된다. Gradient generation 모듈은 keypoint 주변의 local image 영역에 대해서 gradient 값을 생성하게 되고, 생성된 gradient 값의 histogram 을 사용하여, descriptor 를 생성하게 된다.

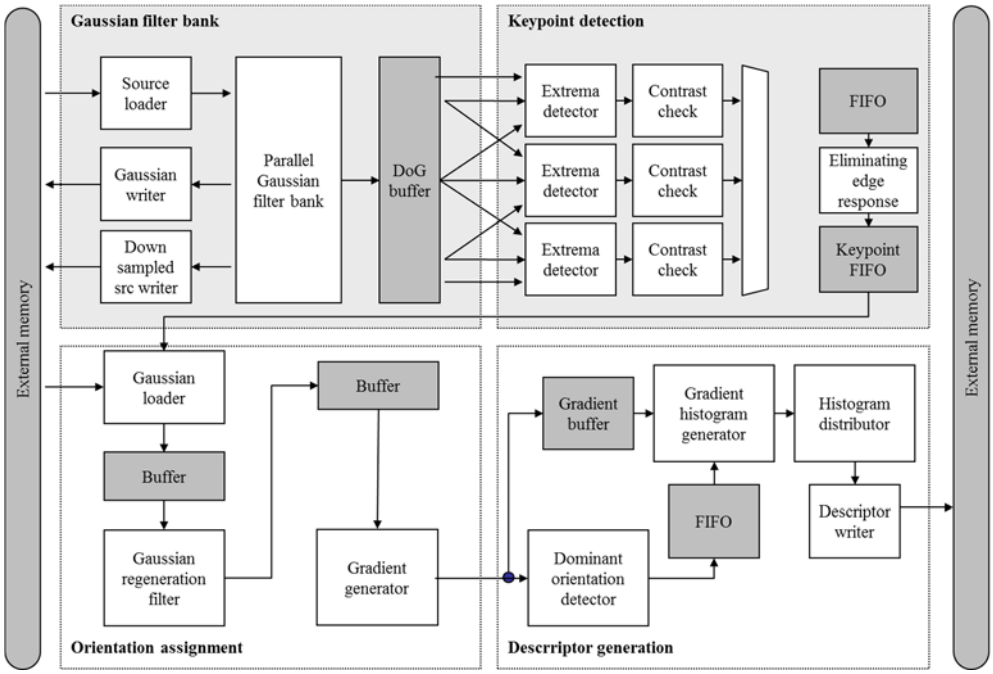


그림 3.8 SIFT Hardware block diagram

이후 절에서는 전체 하드웨어 구조에서 adaptive keypoint 생성을 위한 hardware 구조를 제안한다.

3.4.1 Gaussian filter bank 구조

본 절에서는 기존 SIFT 의 Gaussian filter bank 구조를 설명하고, adaptive keypoint generation 을 위해 제안하는 하드웨어와 기존 Gaussian filter bank 간에 연결 동작에 대해 설명한다.

Gaussian filter bank 는 외부 메모리에 저장된 입력 이미지로부터 Gaussian filtering 된 이미지를 생성하고, 이를 통해 DoG 이미지를 생성하는 부분이다. Gaussian filtering 을 수행하기 위해서, 사용하는 내부 메모리의

양을 줄이기 위해서, Gaussian filter bank 는 블록 단위로 연산이 진행된다.

전체 이미지를 이미지의 폭을 기준으로 96 픽셀 만큼씩 처리하고, 다음 블록을 처리하도록 되어 있다. 그림 3.9은 블록 단위 처리를 위한 연산 순서를 보여준다.

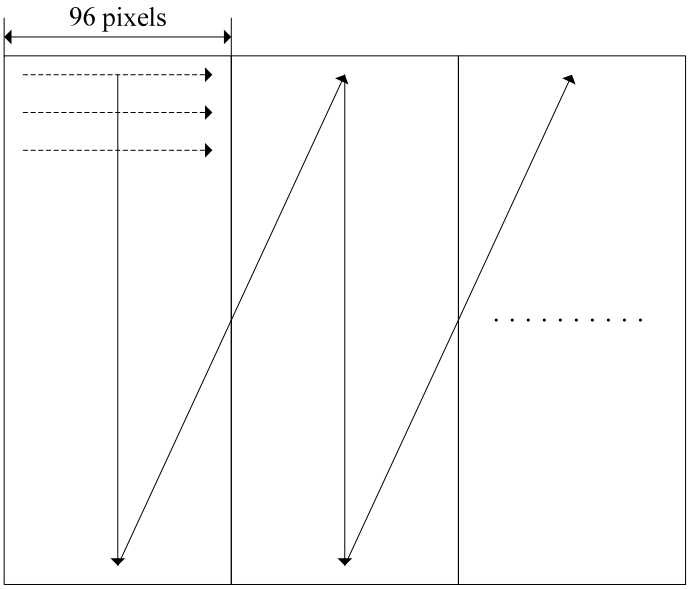


그림 3.9 입력 이미지 data loading 방식

또한 (96 x image height) 의 데이터를 처리하기 위해서 데이터를 한번에 가져오지 않고, circular buffer 형태의 block memory 를 사용하여, 순차적으로 처리하도록 되어 있다. Block memory 의 크기는 그림 3.10와 같이 size 가 128(32 word)이고, 전체의 개수가 64 개인 buffer 를 사용하여 저장한다.

Source loader 로부터 들어오는 픽셀들은 buffer 에 차례로 저장된다. 128 픽셀의 데이터가 다 저장된 이후에는 다음 line 의 128 개 데이터가 buffer 에 저장된다. Source buffer 는 circular 하게 동작하게 된다. 따라서 64 개의

buffer 가 다 채워지면, 가장 먼저 저장된 buffer 는 제일 아래로 내려가고 새로운 128개의 data 가 buffer 에 저장되게 된다.

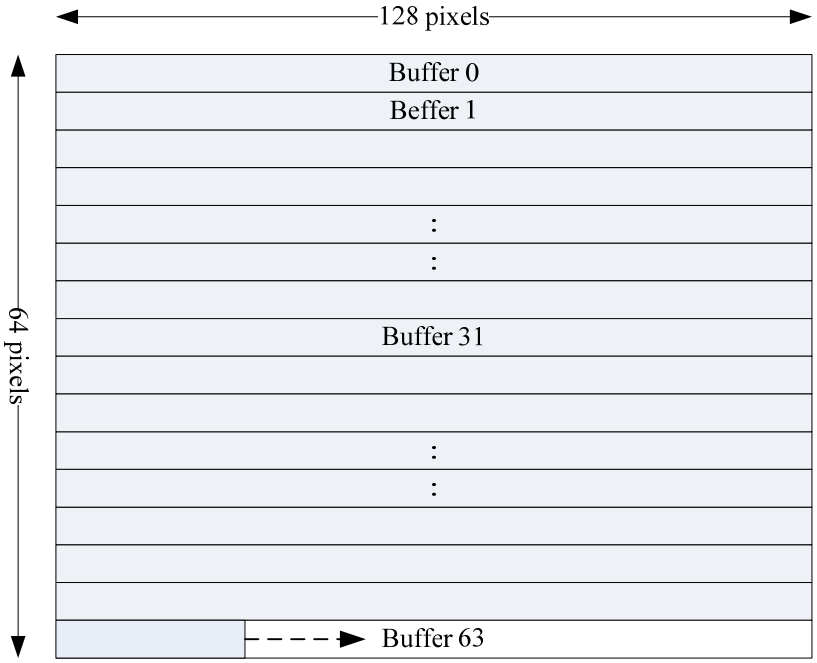


그림 3.10 Source buffer 구조

내부에서 사용하는 circular buffer 의 메모리 크기는 Gaussian filter 의 kernel size 와 관련이 있다. Gaussian filter 는 Gaussian 함수를 FIR(Finite Impulse Response) filter 로 근사화하여 계산한다. Gaussian 함수는 중심점에서 멀어질수록 함수값이 급격히 작아지므로 실용적으로 사용시에는 일정 범위의 filter 계수만을 적용하여 사용하게 된다. 본 연구에서는 함수값이 Gaussian function 최대값의 1% 이하로 떨어지는 지점까지의 함수값을 필터 계수로 사용하였다. 이 지점은 $2\sigma\sqrt{ln10} \approx 3\sigma$ 에 위치하게 된다. Gaussian

함수는 좌우 대칭을 이루기 때문에, Gaussian kernel 의 크기는 $Size_{guassian} = 2 \cdot Round(3\sigma) + 1$ 로 계산된다[4]. 본 연구에서는 $\sigma_0 = 1.6$ 의 값을 사용하였다. 따라서 scale-space 를 구성하기 위해서, filter scale 에 따른 Gaussian kernel 크기는 표 3.3 와 같이 만들어진다.

표 3.3 Filter scale 에 따른 Gaussian kernel 크기

Filter scale	σ_0	$\sigma_0 \cdot 2^{1/3}$	$\sigma_0 \cdot 2^{2/3}$	$\sigma_0 \cdot 2$	$\sigma_0 \cdot 2^{4/3}$	$\sigma_0 \cdot 2^{5/3}$
Gaussian Kernel size	11	13	17	21	25	31

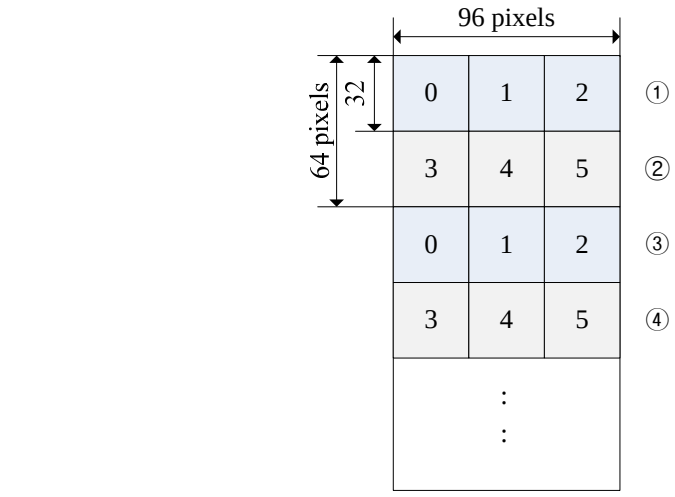
Gaussian filter bank module 은 가로 방향과 세로 방향에 대해서 각각 Gaussian filtering 을 수행한다. 따라서 source line buffer 의 크기는 SIFT 동작을 위해서, 가장 큰 Gaussian kernel 의 Size 를 기준으로 최소 31 line 이상의 Data 가 필요하게 된다. 따라서 31 개의 line buffer 가 다 채워지게 되면, 세로로 한 픽셀을 읽어 Gaussian filter bank 에 보내게 된다.

또한 가장 큰 kernel size 를 기준으로 96 개의 픽셀을 filtering 을 하기 위해서 위해서 좌, 우 15 픽셀의 data 가 추가로 필요하고, data 를 word 단위로 처리하기 때문에, 좌, 우 16 픽셀에 해당하는 크기만큼의 data 를 추가로 저장하게 된다. 따라서 한 line buffer 의 크기는 128 로 설정한다.

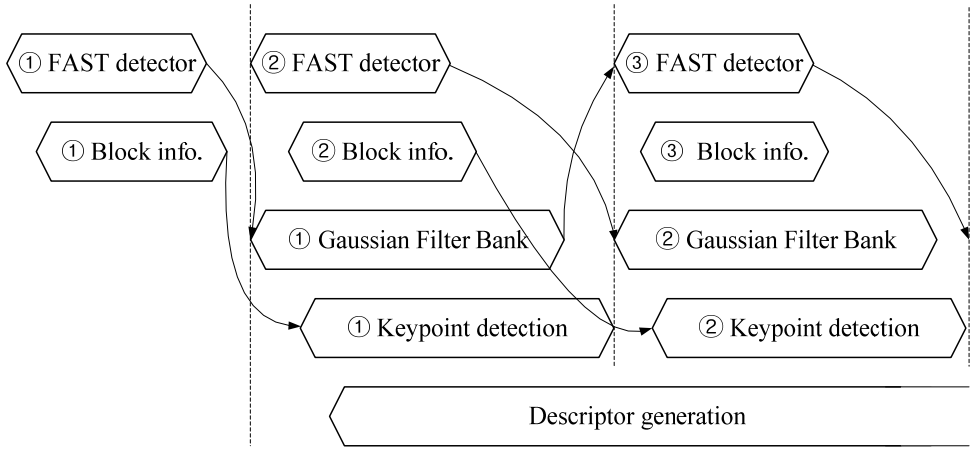
제안한 hardware 구조에서는 해당 픽셀의 SIFT 의 연산 이전에 keypoint 의 일정 영역 내의 발생 분포를 예측하기 위해서, 추가적으로 32 line 의 data 를 source line buffer 에 저장하여 사용하였다.

본 연구에서는 32x32 단위를 keypoint 분포를 예측하기 위한 단위로 사용하였다. SIFT 는 (96 x image height) 단위로 나누어 연산이 진행되므로 그림 3.11 (a) 와 같이 source buffer 에는 6 개의 keypoint 를 예측하기 위한 단위가 만들어지게 된다. 따라서, 세로 방향의 Gaussian filtering 을 위해서 필요한 buffer 의 line 수는 32 이지만 미리 keypoint 분포를 예측하기 위해서, 그림 3.10 에서와 같이 circular buffer 구조의 크기를 64 line 으로 구성하였다.

Keypoint 분포를 예측하기 위한 FAST detector 와 Gaussian filter bank 는 그림 3.11 (b) 와 같이 시간적으로 FAST detector 가 먼저 수행되어 ① 영역에 대한 keypoint 분포를 예측한 이후에 그림 3.10(b) 와 같이 ① 영역의 Gaussian filtering 을 수행하도록 되어 있다. FAST detector 는 Gaussian filter bank 의 수행시간보다 빠르게 수행되기 때문에, source line buffer 에서 data 를 읽어오는 기준은 SIFT 의 Gaussian filtering 의 연산속도에 영향을 받게 된다. 또한 FAST detector 와 Gaussian filter bank 는 handshaking 구조로 동작하기 때문에, 그림에서와 같이 FAST 는 ② 번 영역에 대한 FAST 의 연산이 완료되면, Gaussian filter bank 에 이를 알려주고, ③ 번 영역의 계산을 수행하고, Gaussian filter bank 는 ② 번 영역에 대한 Gaussian filtering 을 수행하게 된다. 따라서 6 개의 블록 정보를 저장할 공간만 있으면, 순차적으로 사용이 가능하다.



(a)



(b)

그림 3.11 FAST 와 Gaussian filter bank 연동 구조 (a) Keypoint 예측 블록 구성 (b) Timing diagram

위와 같은 buffer 구조를 통해 순차적으로 keypoint 분포 예측과 Gaussian filtering 이 가능하게 되며, 전체 수행시간 측면에서 SIFT 만 수행

하는 하드웨어 구조에서 초기 3개 블록에 대한 keypoint 발생을 예측하기 위한 FAST detector 의 연산에 의한 latency 만 가지게 된다.

3.4.2 FAST detector

블록내의 이미지 특성을 파악하기 위해서 본 연구에서는 FAST detector 에서 발생하는 keypoint 의 분포를 사용한다. 이를 위해 FAST hardware 의 동작에 대해서 설명한다. FAST detector 는 하나의 픽셀에 대한 keypoint 여부를 조사하기 위해서 7x7 윈도우 크기의 값을 필요로 한다. 또한 non-maximal suppression 을 통해 최종적으로 keypoint 를 판단하기 위해서 현재 keypoint 를 중심으로 3x3 범위내에서 발생된 keypoint 의 score 값을 사용한다.

따라서 한 픽셀을 FAST keypoint 로 최종 판단하기 위해선 입력 이미지 기준으로 9x9 의 주변 픽셀값을 필요로 한다. 따라서, 7x9 단위의 블록이 raster-scan order 로 데이터를 처리하게 된다. 7x9 윈도우를 사용하게 되면, non-maximal suppression 과정을 수행시에 필요한 위, 아래의 keypoint 에 대한 score 값을 가지고 있기 때문에, non-maximal suppression 을 동시에 수행 가능하다

7x9 윈도우는 그림 3.12 과 같이 3개의 위치에 대한 corner 여부를 동시에 조사할 수 있다. 점선에 해당하는 픽셀들은 ‘x’ 로 표시된 위치를 corner 로 판단하기 위해서 필요한 주변 픽셀들을 표시한다.

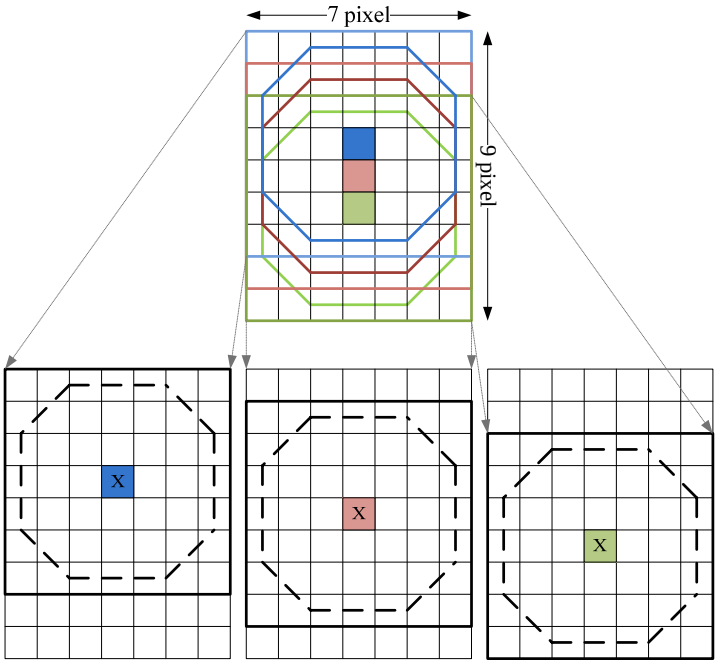


그림 3.12 7x9 window corner detection

따라서 FAST detector 를 위한 하드웨어 구성은 그림 3.13 와 같이 3 개의 corner detection 과 score 를 계산하는 부분과, non-maximal suppression 을 처리하는 부분으로 구성된다.

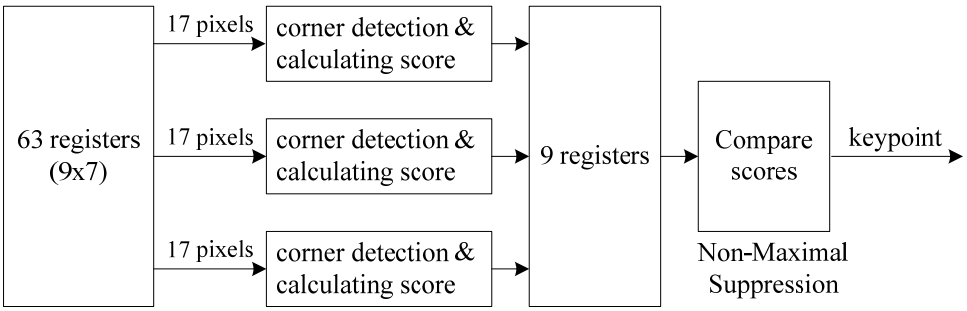


그림 3.13 FAST detector 하드웨어 구조

3.5 하드웨어 구조

그림 3.14 은 이전 절에서 설명한 Gaussian filter bank 구조와 FAST detector 를 적용한, adaptive keypoint generation 을 위한 하드웨어 구조를 보여준다.

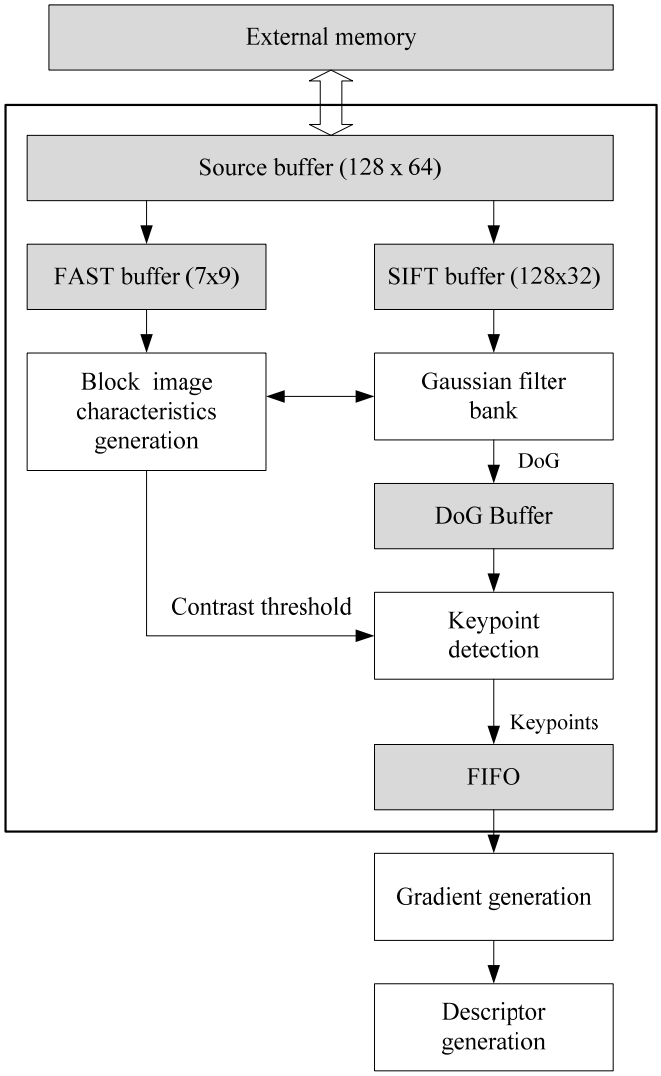


그림 3.14 Adaptive keypoint 생성을 위한 hardware 구조

전체 시스템은 외부 메모리에 저장된 데이터를 읽어서 source buffer 에 저장하는 ‘Source loader’ 부분, 저장된 source buffer 로 부터 블록내의 이미지의 characteristics 를 예측하는 ‘Block image characteristics generation’ 부분, keypoint 를 생성하기 위한 ‘Gaussian filter bank’, ‘Keypoint detection’ 부분, 생성된 keypoint 로부터 gradient 를 생성하는 ‘Gradient generation’ 부분과 ‘Descriptor generation’ 부분으로 이루어져 있다. 이중 adaptive keypoint generation 을 위해선 점선 부분으로 표시된 영역에 블록 이미지의 복잡도를 예측하는 부분을 추가하고, 이를 통해 keypoint 생성단계에서, keypoint 조절이 가능한 구조를 적용하였다.

3.5.1 Block image characteristics generation

그림 3.15 는 'Block image characteristics generation' 부분에 대한 하드웨어 구조를 보여 준다.

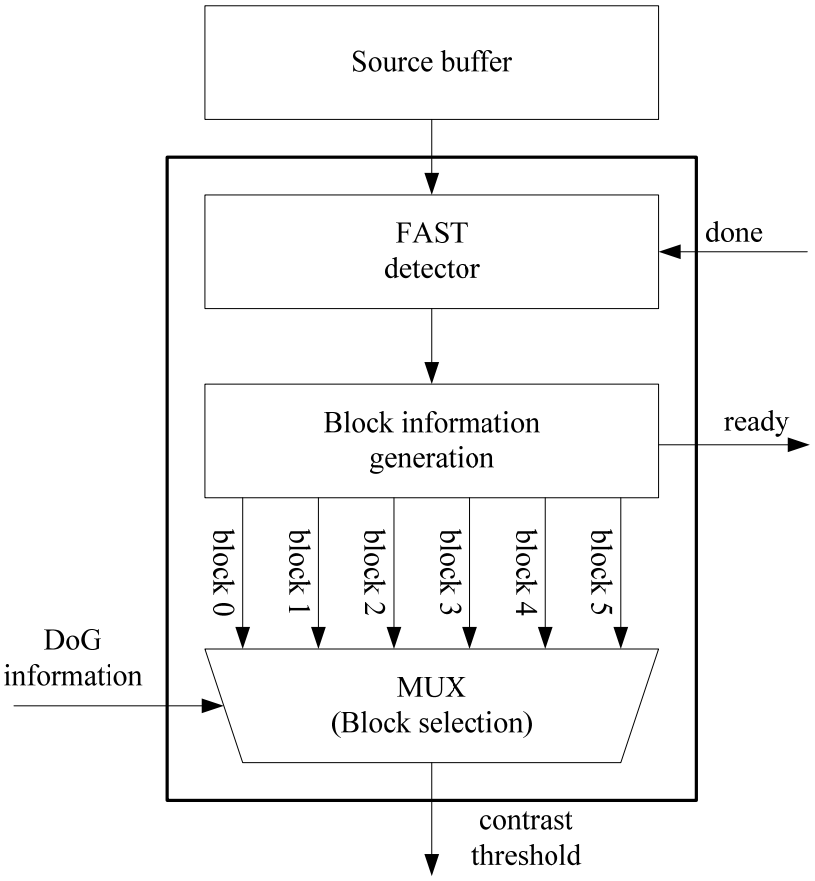


그림 3.15 Block image characteristics generation

'Block image characteristics generation' 부분은 입력으로 source loader 를 사용하여 저장한 source buffer 의 data 를 받고, 블록 단위의 이미지의 복잡도 정보 생성이 끝나는 시점에 'Gaussian filter bank' 에 해당 블록의

Gaussian filter bank 의 시작이 가능함을 알려 주도록 되어 있다. 또한 생성된 블록 단위의 이미지의 복잡도 정보를 사용하여, keypoint detector 모듈이 수행되는 부분에 맞는 contrast threshold 를 전달해 주도록 되어 있다. 블록 이미지의 복잡도가 생성되는 시점은 해당 위치의 contrast threshold 가 동작되는 시점과 다르게 되기 때문에, 이미 저장된 블록 단위의 정보로부터 keypoint detector 가 DoG data 를 읽어가는 순서에 맞는 contrast 값을 제공해 주는 과정이 필요하게 된다. 블록별 이미지의 복잡도 정보는 블록내에 발생한 FAST keypoint 수의 합을 사용하여 결정한다. 또한 이를 통해 표 3.4 와 같이 이미지의 복잡도를 판단하고 contrast threshold 값을 조정하게 된다. 이때의 threshold 값은 적용되는 application에 따라서 조절 가능하다. FAST keypoint 의 발생 분포가 높은 블록은 '복잡'으로 판단하고, 이 블록은 contrast threshold 의 값을 증가한다. 반대로 '단순'으로 판단된 영역에서는 contrast threshold 를 높일 경우 keypoint 가 없어질 가능성이 있으므로 contrast threshold 값을 유지한다.

표 3.4 블록별 이미지 복잡도에 따른 contrast threshold

	블록내 FAST keypoint 수	블록내 이미지 복잡도	Contrast threshold
I	0	-	-
II	# of FAST KP < th	단순	유지
III	# of FAST KP > th	복잡	증가

SIFT keypoint detector 단계에서 적용되는 contrast threshold 값은 SIFT keypoint detector 에 전달되는 DoG 이미지 위치에 따라서 달라지게 된다. 따라서, Block image characteristics generation 에서는 입력되는 DoG 위치 정보를 통해서 그에 맞는 블록 정보를 발생하도록 되어 있다.

3.5.2 Gaussian filter bank

그림 3.16 Gaussian filter bank 구조를 보여준다. Gaussian filter bank 는 입력 이미지로부터 DoG 이미지를 생성하는 부분이다. 이때, Gaussian filter bank 에서 사용하는 입력 이미지가 저장된 source buffer 는 앞 절의 'Block image characteristics generation' 에서 사용하는 공간과 동일한 공간이다. 3 개의 line buffer 는 각각 하나의 octave 에 대한 source image 의 line 을 저장한다. Octave 1 와 octave 2 에 대한 연산에서 사용하는 입력 이미지는 현재 octave 의 (S+1) 번째 Gaussian-blurred image $L_s(x,y)$ 을 1/2씩 down-sampling 하여 얻을 수 있다. Octave 1, Octave 2 를 위해 사용하는 down-sampling 된 이미지는 외부 메모리에 저장되어, 해당 Octave 계산시에 외부로부터 가져와서 사용하게 된다. 이와 같은 Gaussian filter bank 를 동작은 'Block image characteristics generation' 에서 생성되는 ready 신호에 따라서 동작하게 된다. 즉, Gaussian filtering 을 수행하는 픽셀에 대한 이미지의 복잡도 정보가 생성된 이후에 동작하도록 되어 있다.

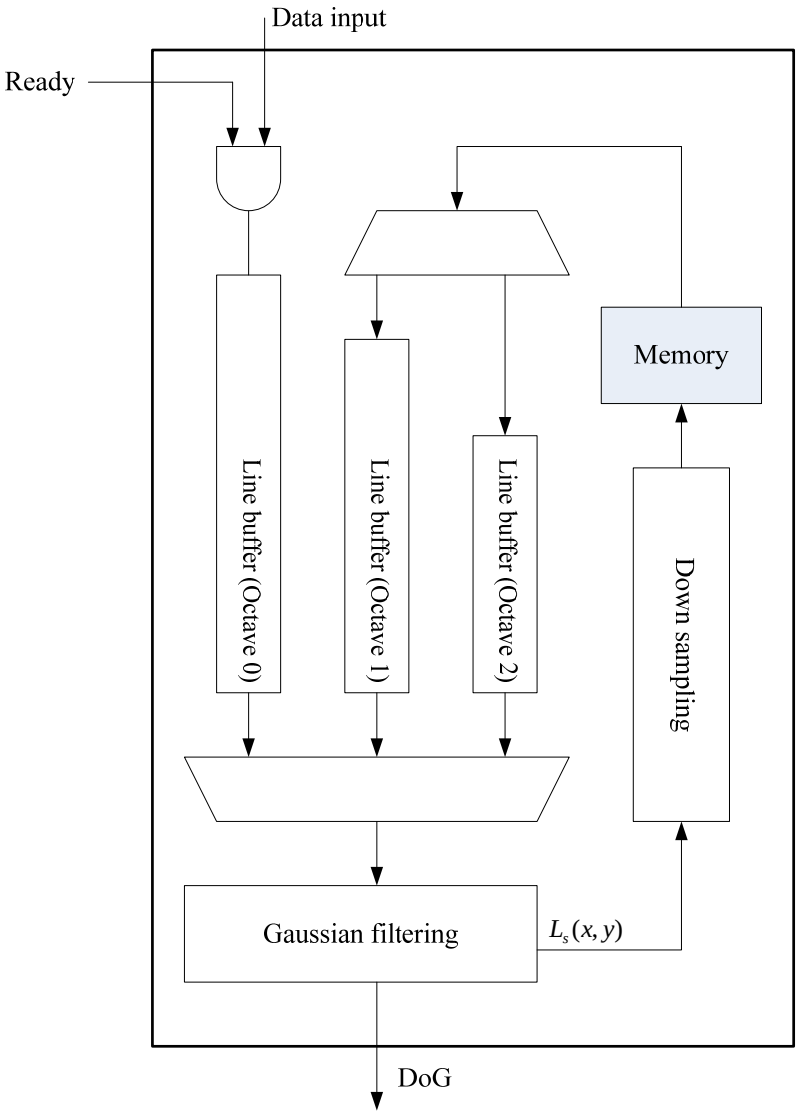


그림 3.16 Gaussian filter bank

3.5.3 Keypoint detector

그림 3.17 은 keypoint detection 을 위한 하드웨어 구조를 보여 준다. 본 연구에서는 5개의 DoG 이미지를 생성한다. 이를 통해 3 개의 extrema detector 를 위한 조합이 가능하게 된다. 속도를 향상하기 위해서, 하드웨어에서는 3개의 extrema detector 가 병렬적으로 동작하는 구조를 가지고 있다. 검출된 local extrema 는 contrast check 단계를 거치게 되는데, 이때, 'Block image characteristics generation' 의 contrast threshold 값이 사용된다. 이후에 이루어지는 'Eliminating edge response' 모듈은 검출된 keypoint 후보군에 대해서만 연산이 이루어지기 때문에, 많은 연산량을 필요로 하지 않는다. 또한 각각의 extrema detector 와 contrast check 에 의해서 생성되는 keypoint 는 서로 다른 개수가 생성된다. 따라서, Eliminating edge response 을 여러 개로 구성할 경우, load imbalance 하기 때문에, 병렬화로 인한 효과를 극대화 하기 어렵게 된다. 또한, 생성된 keypoint 는 descriptor 계산을 위해서 저장된다

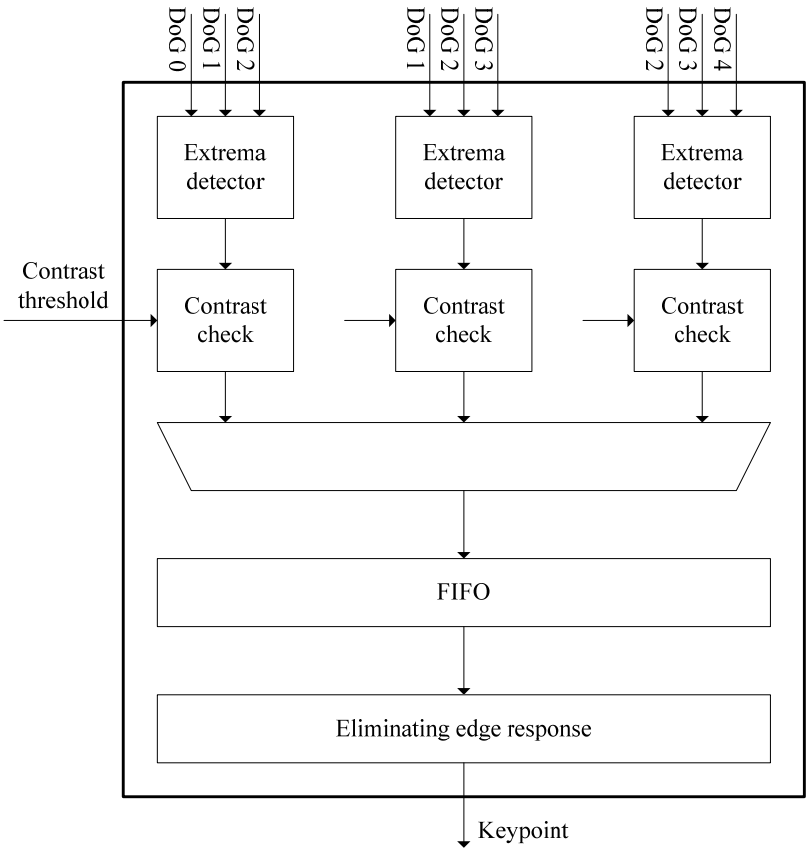


그림 3.17 Keypoint detection

3.6 성능 평가

이번 장에서 제안된 하드웨어 구조는 이미지에서 발생하는 keypoint 를 블록별로 adaptive 하게 조절할 수 있는 방법을 제공하고 있다.

제안된 방법은 이미지내에 SIFT keypoint 발생 분포가 높은 영역과 keypoint 발생 분포가 낮은 영역을 SIFT keypoint detector 동작 이전에 예측하여, SIFT 에서 발생하는 keypoint 의 수를 조절할 수 있도록 되어 있다. 또한 keypoint 발생 분포를 예측하기 위한 부분은 Gaussian filter bank 동작에 비해서 빠르게 수행 가능하여, 하드웨어로 구성된 SIFT 의 동작 구조에 영향을 미치지 않고 병렬로 수행이 가능하다.

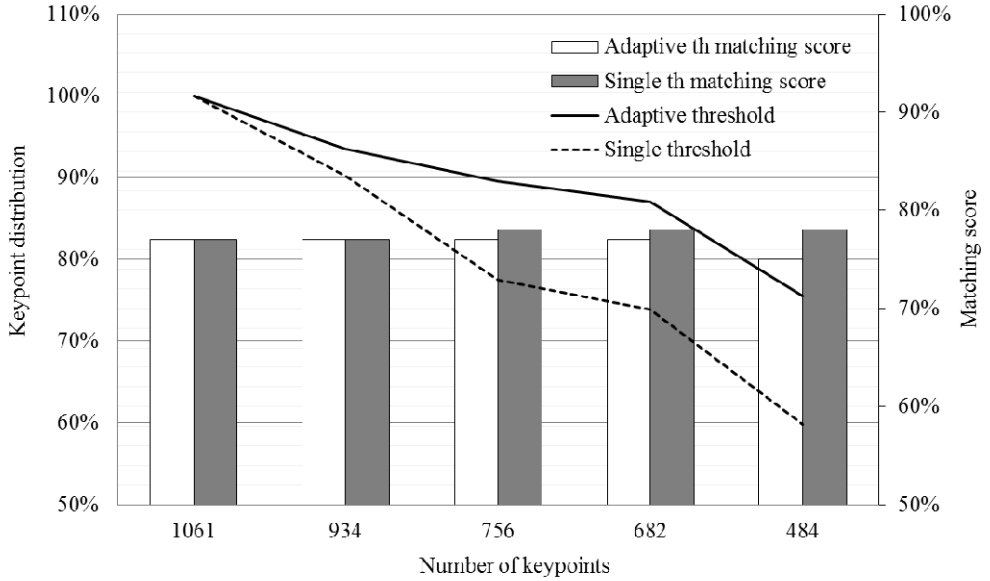
3.6.1 성능 실험 결과

그림 3.18 은 Graffiti 이미지를 사용해서 adaptive threshold control 방법에 의해 keypoint 수를 조절하였을 경우와 전체 이미지에 대해서 single threshold 를 적용하였을 경우에 대한 keypoint 분포 변화를 보여준다.

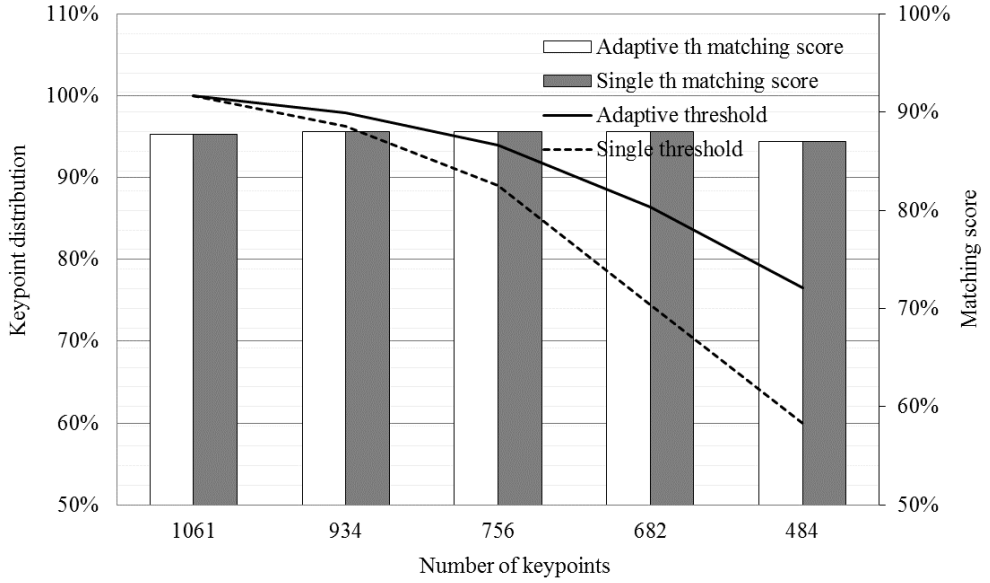
그림 3.18 의 꺾은선 그래프는 Lowe[9] 논문에서 사용한 0.03 contrast threshold 를 사용했을 때 발생하는 keypoint 분포를 기준으로 keypoint 수를 줄였을때의 분포를 상대적으로 표시한다. 또한 그림 3.18 의 막대형 그래프는 각각의 방법에서의 matching score 를 보여준다. 그림 3.18 에서 보이는 바와 같이 adaptive threshold 방법을 적용하였을 경우 keypoint 의 분포가 상대적으로 더 고르게 분포함을 알 수 있다.

그림 3.19 는 (b) 의 초기 SIFT keypoint 분포로부터 single threshold 값을 사용해서, keypoint 의 수를 줄였을 경우와 adaptive threshold 값을 적용

해서, keypoint 를 줄였을 경우에 대한 keypoint 분포를 보여준다. 그림 3.19 (c), (e) 는 초기의 keypoint 로부터 single threshold 값을 조절하여, 756, 484 개로 keypoint 수를 줄였을 경우에 대한 분포를 보여 주고, 그림 3.19 (d), (f) 는 adaptive threshold 를 적용하였을때의 결과를 보여준다. Adaptive threshold 를 적용하였을 경우에서 keypoint 의 분포가 더 넓게 분포 됨을 알 수 있다.



(a)



(b)

그림 3.18 keypoint 수에 따른 keypoint 분포 (a) Graffiti (b) Boat

표 3.5 는 여러 종류의 입력 이미지에 대해 keypoint 를 조절했을때 실험 결과를 보여준다. 실험은 0.03 의 contrast threshold 를 사용했을 때 생성되는 keypoint 수의 50% keypoint 를 발생했을 때를 기준으로, single threshold 를 사용했을 경우와 adaptive threshold 를 사용했을 때를 비교하였다. 이전에 실험결과에서 살펴본 바와 같이 다른 이미지들에 대해서도 adaptive threshold 를 사용한 결과가 10~15% 정도 keypoint 의 분포가 넓게 생성됨을 볼 수 있다. 또한 전체적으로 이때의 matching score 는 single threshold 나 adaptive threshold 를 사용한 경우 비슷한 결과를 보여준다.

표 3.5 입력 이미지에 따른 실험 결과

		Graffiti	Bikes	Boat	Leuven
Single threshold	Distribution	60%	70%	60%	57%
	Matching score	78%	90%	87%	90%
Adaptive threshold	Distribution	76%	83%	76%	67%
	Matching score	75%	90%	84%	89%

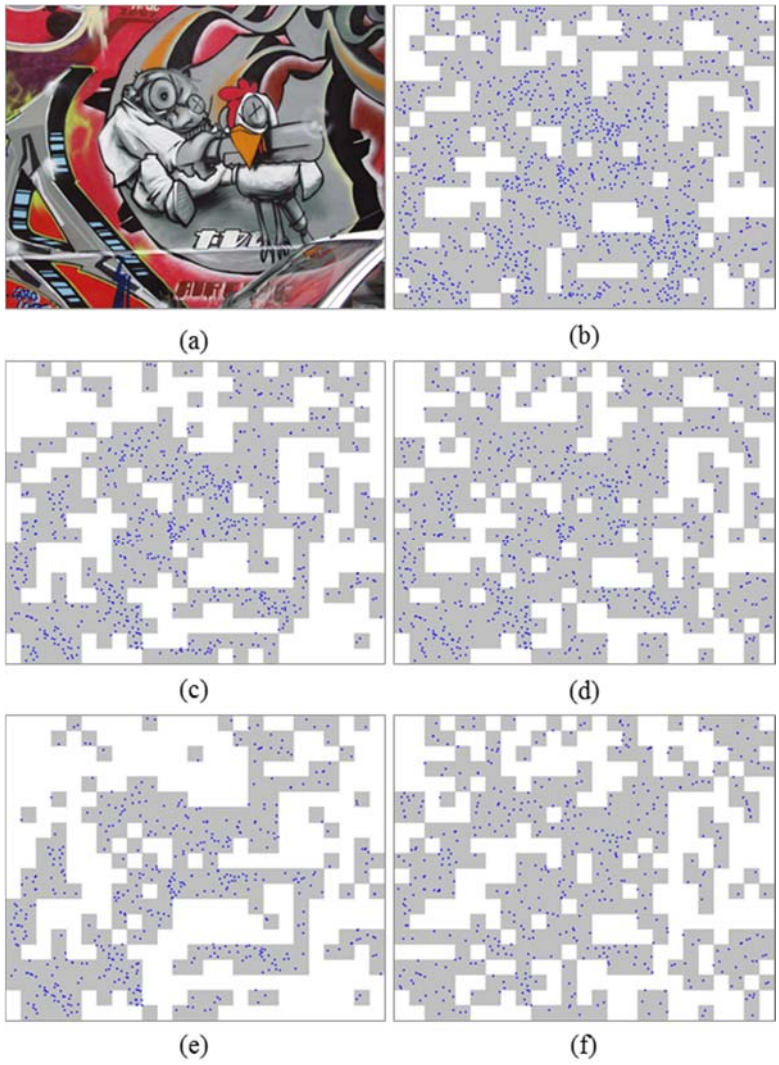
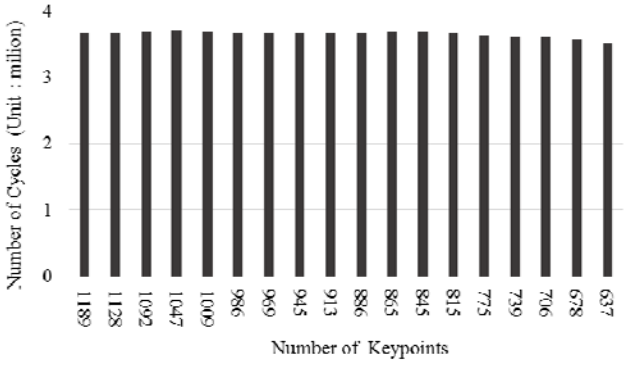


그림 3.19 Threshold 에 따른 keypoint 분포 (a) 실험 이미지 (b) 초기 keypoint 분포(n=1061) (c) Single threshold (n=756) (d) Adaptive threshold(n=756) (e) Single threshold (n=484) (f) Adaptive threshold(n=484)

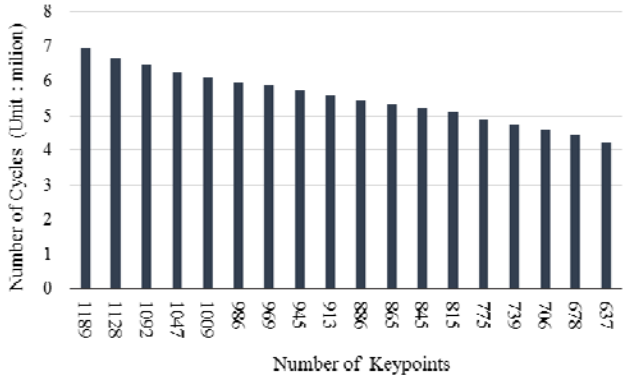
3.6.2 하드웨어 실험 결과

Adaptive keypoint 생성에 따른 keypoint detector 와 descriptor generation 의 수행 시간을 살펴보면 그림 3.20 (a) 와 같이 keypoint 수가 증가함에 따라서, keypoint detection 의 경우엔 일정한 수행 시간을 보이지만, descriptor generation 은 수행 시간이 줄어드는 결과를 볼 수 있다.

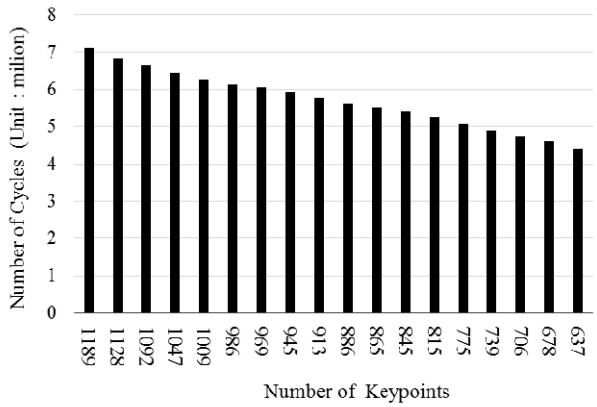
또한 전체적인 수행시간은 keypoint detector 와 descriptor generation 이 병렬로 진행되므로, keypoint detector 와 descriptor generation 수행시간의 합이 아닌 그림 3.20 (c) 와 같이 나타나고, 수행시간이 줄어드는 비율은 발생된 keypoint 수의 감소에 따라 descriptor generation 이 줄어드는 만큼 줄어들게 된다. 또한 전체적인 performance 관점에서, descriptor generation 부분이 keypoint detection 부분보다 더 많은 hardware 수행 과정을 필요로 하는 것을 볼 수 있다.



(a)



(b)



(c)

그림 3.20 Keypoint 수에 따른 하드웨어 수행시간 (a) Keypoint detection (b) descriptor generation (c) 전체 SIFT 수행시간

표 3.2 는 전체 하드웨어에 대한 합성 결과를 보여 준다. 합성 툴은 Synopsys 사의 Design compiler 를 사용하였다. Gate count 계산을 위해 target frequency 는 50Mhz 로 설정하고, silterra 0.13um library 를 사용하였다. 실험 결과는 SIFT 하드웨어에 adaptive keypoint 생성을 위해 추가된 'Block image characteristics generation' 부분이 전체 하드웨어의 gate count 에 비해 약 5 % 정도의 추가 logic 이 발생함을 보여 준다.

표 3.6 Adaptive keypoint generation 하드웨어 합성 결과

Module Name	Gate count	상대 비율
Space-space generation	478003	73.1%
Block image characteristics generation	**32526**	**5%**
Keypoint detection	49439	7.6%
Descriptor generation	94126	14.4%
Total	654094	100%

또한 제안된 하드웨어 에서는 초기 (96x32) 크기의 블록 이미지의 특성을 분석하기 위해서, 초기 latency 를 가지고 있다. 실험에서는 5932 cycle 이 초기의 latency 로 측정되었다. 초기의 latency 이후의 과정에서는 SIFT 의 Gaussian filter bank 블록과 블록내의 이미지 복잡도를 분석하기 위한 하드웨어가 병렬적으로 동작하기 때문에, 이로 인한 추가적인 수행시간의 증가는 발생하지 않는다. 이는 'Block image characteristics generation' 부분의 하드웨어 수행시간이 Gaussian filter bank 수행시간에 비해 빠르게 수행되므로

가능하다.

제안된 방법은 SIFT 하드웨어의 pipeline 구조가 깨지거나, 추가 하드웨어에서 외부와의 bus traffic 로 인한 delay 가 발생하지 않도록 설계되어 있다. 또한 블록단위로 수행하는 SIFT 의 내부메모리를 공유하는 구조로 설계가 되어 있다. SIFT 하드웨어는 블록 기반의 연산을 수행하고 있다. 따라서 제안한 방법은 adaptive 한 keypoint 조절을 위해서, 마찬가지로 블록 단위로 복잡도를 예측 가능하도록 되어 있다.

제4장 Region-constrained feature matching

서로 다른 이미지에 존재하는 물체를 인식하기 위한 방법으로 local feature 에 기반한 matching 방법이 사용되고 있다. 본 장에서는 여러 가지 feature matching 방법 중, hierarchical agglomerative clustering 에 기반한 matching 방법의 특징을 살펴보고, 이를 통해 inlier 와 outlier 를 효과적으로 구분하는 방법을 제안한다.

4.1 Hierarchical agglomerative clustering

Feature matching 을 수행하기 위해서, Euclidean distance 을 사용하여, descriptor 간에 유사성을 비교하는 방법은, local patch 의 유사성만을 비교하기 때문에, 부분적으로 유사한 다른 부분의 descriptor 로 correspondence 가 이루어 질 수 있다. 이는 많은 수의 잘못된 correspondence 가 포함될 수 있음을 의미하므로 descriptor 간의 유사성을 사용한 초기의 correspondence 로부터 correct correspondence(inlier) 와 incorrect correspondence(outlier) 를 구분하는 방법이 필요하다.

이전의 방법들 중 RANSAC 방법을 사용하여 affine transform model 을 추정하거나[9][11], geometric 한 정보를 사용하여, 이미지내의 keypoint 간 distance 나 angle 이 상대적으로 불변함을 이용하는 방법은[30][31][32] non-rigid 이미지에는 효과적이지 못하거나, 많은 수의 correspondence 가

존재하는 경우, 잘못된 keypoint 의 조합에 의해서도 distance 나 angle 간의 불변한 성질이 유지될 가능성이 높은 문제가 발생하게 된다. 따라서, 최근에 feature 간의 clustering 을 이용하여 신뢰성 높은 feature set 을 얻는 방법들이 있다[33]. 이와 같은 clustering 방법들은 이미지 내의 모든 feature correspondence 를 대상으로 모든 inter-cluster similarity 가 intra-cluster similarity 보다 클 때까지 반복적으로 수행한다.

그림 4.1 는 찾고자 하는 대상 이미지 그림 4.1 (a) 에 대해서 이미지가 변형이 되었을때 RANSAC 방법과 clustering 방법에 의해 matching 을 수행한 결과를 각각 그림 4.1(b), (c) 에 보여준다. 결과에서 볼 수 있듯이, 이미지의 변형이 심한 경우 RANSAC 방법으로는 object 를 정확히 찾기 어렵지만, clustering 의 경우엔 보다 정확한 결과를 얻을 수 있다.

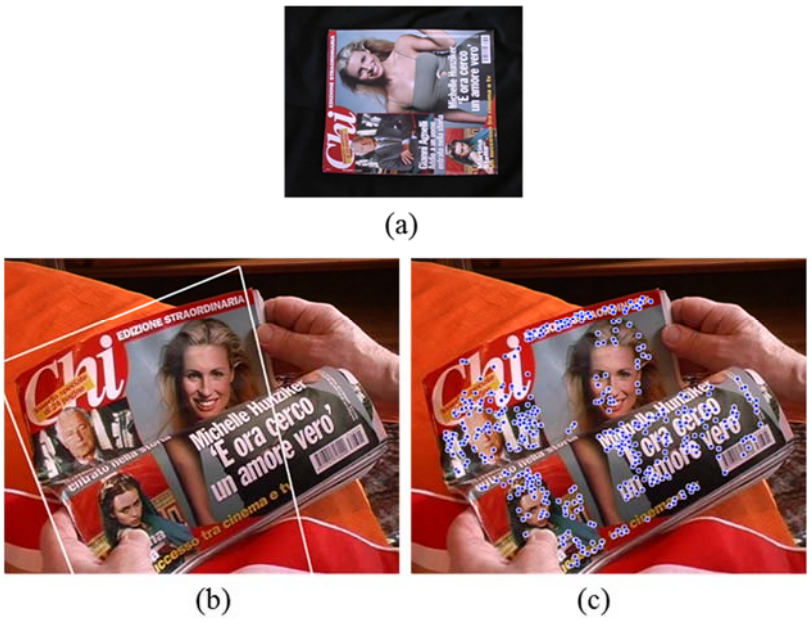


그림 4.1 Matching results (a) Model (b) RANSAC (c) Clustering

본 연구에서는 HAC(Hierarchical Agglomerative Clustering) 기반의 clustering 알고리즘을 사용한다[35][38]. 그림 4.2은 일반적인 clustering 방법을 보여준다. 첫 번째 단계는 (Correspondence extraction) 은 서로 다른 이미지에 존재하는 descriptor 간의 유사성으로 계산한 local feature 간의 초기 correspondence 를 계산하는 부분이다. 두 번째 단계는 (Cluster similarity) correspondence 들간의 similarity 를 계산하는 부분이다. 이와 같은 similarity 는 다음 단계(Clustering) 에서 clustering 을 위한 조건으로 사용된다. 두 번째 단계와 세 번째 단계는 모든 inter-cluster similarity 가 intra-cluster similarity 보다 클 때까지 반복적으로 수행된다.

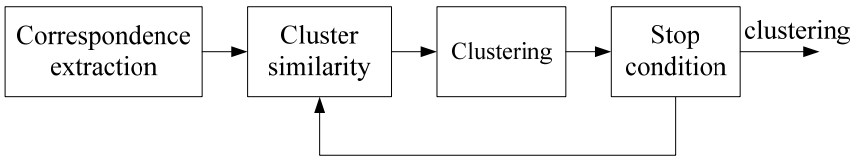


그림 4.2 A general flow of a clustering algorithm

HAC 는 clustering 방법 중 하나로 그림 4.2 과 같은 flow 를 가지고 있다. HAC 에 대한 전체 과정은 다음과 같다.

HAC Algorithm

Step 1: Determine all inter-correspondence similarities

Step 2: Select two closest correspondences or clusters and form a cluster

Step 3: Redefine similarities between the new cluster generated in Step 2 and the other correspondences or clusters

Step 4: Return to Step 2 until inter-cluster similarity is larger than intra-cluster similarity

Geometric similarity 를 계산하기 위해서, 먼저 두 개의 correspondence m_i , m_j 간의 distance 를 정의한다. 이때, p 와 q 을 각각 다른 이미지에 속해 있는 keypoint 라고 하고, 두 keypoint 는 h_i 에 의해서 correspondence 가 이루어져 있다고 하자. x_i 과 x_i' 가 각각 p 와 q 의 위치를 나타낸다고 하면, p 와 q 사이의 correspondence 는 $m_i = (x_i, x_i', h_i)$과 같이 표현할 수 있다. 이때 두 개의 correspondence $m_i = (x_i, x_i', h_i)$ 과 $m_j = (x_j, x_j', h_j)$ 사이의 distance 는 다음과 같이 정의한다[33].

$$
\begin{aligned}
d(m_i, m_j) &= \frac{1}{2}(d(m_j|m_i) + d(m_i|m_j)) \\
d(m_j|m_i) &= \frac{1}{2}(|x_j' - h_i x_j| + |x_j - h_i^{-1} x_j'|) \\
d(m_i|m_j) &= \frac{1}{2}(|x_i' - h_j x_i| + |x_i - h_j^{-1} x_i'|)
\end{aligned}
\tag{4.1}
$$

여기서 $|\cdot|$ 는 Euclidean distance 를 의미한다. 이와 같은 distance 에 대한 정의를 가지고, G 와 H 로 표현되는 두 개의 cluster 간의 similarity 는 식 (4.2) 와 같이 두 cluster 사이의 가장 가까운 correspondence 간의 거리로 정의된다.

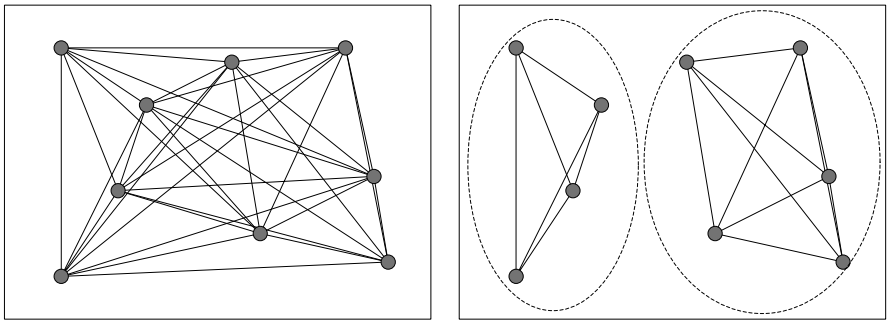
$$D(G,H) = \min_{\forall m_i \in G,\, \forall m_j \in H} d(m_i, m_j) \tag{4.2}$$

위의 정의를 통한 HAC 방법은 clustering 을 통해서, 효과적으로 inlier 와 outlier 간의 구분이 가능하지만, correspondence 의 수가 증가함에 따라 연산해야 할 데이터가 기하급수적으로 늘어나기 때문에, 급격하게 수행시간 증가가 발생하게 된다. 따라서 본 연구에서는 matching 의 정확도의 감소 없이 computational complexity 를 줄이기 위한 연구를 진행하였다.

4.2 Region-constrained clustering

본 연구에서는 효과적으로 clustering 을 수행하기 위해서, clustering 을 위한 candidate region 을 설정하여, candidate region 별로 clustering 을 수행하고, 그것들의 집합으로써 결과를 구성하는 방법을 사용하였다[34].

이번 절에서는 유사한 속성들로 이루어진 candidate region 을 생성하는 방법과, candidate region 내에 keypoint 를 가지고 clustering 을 수행하는 region-constrained clustering 방법에 대해서 설명한다.



(a) (b)

그림 4.3 Clustering 방법 비교(a) HAC (b) Region-constrained clustering

그림 4.3 는 HAC 방법과 region-constrained clustering 방법의 차이점을 보여준다. 그림 4.3 (b) 에 보이는 점선으로된 타원은 candidate region 을 표시한다. 두 가지의 방법은 동일한 keypoint 를 대상으로 하고 있다. 그러나 region-constrained clustering 방법은 candidate region 별로 candidate region 내의 keypoint 에 대해서만 clustering 을 수행하게 된다.

4.2.1 Geometric relationship in correspondence

P, Q 를 각각 두 개의 이미지로부터 생성된 keypoint 의 집합이라고 할 때, 두 개의 feature set P, Q 내의 feature vector p_u, q_v 는 식 (4.3) 과 같이 표현이 가능하다.

$$\begin{aligned} p_u &= [(x_u, y_u), \sigma_u, \theta_u, f_u] \\ q_v &= [(x_v, y_v), \sigma_v, \theta_v, f_v] \end{aligned} \tag{4.3}$$

여기서 (x_u, y_u) 는 keypoint 위치에서의 coordinate 를 의미한다. 또한 σ_u 와 θ_u 는 해당 위치에서 scale, orientation 정보를 나타낸다. f_u 는 descriptor 를 의미한다. 초기 correspondence 는 descriptor 들 간에 가장 가까운 Euclidean distance 와 두 번째로 가까운 Euclidean distance 를 비교하여, 계산된다. 이와 같은 초기 correspondence 를 가지고, correspondence 간의 homography matrix 를 계산할 수 있다. 각각의 feature vector 는 scale 과 orientation 에 대한 정보를 가지고 있다. 따라서 correspondence 에 대한 homography matrix (h)는 식 (4.4) 와 같이 scale 과 rotation, translation 정보를 가진 matrix 들의 곱으로 표현이 가능하다.

$$\begin{aligned} h &= [S(\Delta\sigma)][R(\Delta\theta)][T(\Delta x, \Delta y)] \\ &= \begin{pmatrix} \Delta\sigma & 0 & 0 \\ 0 & \Delta\sigma & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} cos\Delta\theta & -sin\Delta\theta & 0 \\ sin\Delta\theta & cos\Delta\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{4.4}$$

여기서 $\Delta\sigma = \sigma_v/\sigma_u$, $\Delta\theta = \theta_v - \theta_u$ 와 같이 정의되고, Δx, Δy 는 식 (4.5) 와 같이 표현이 가능하다.

$$\Delta x = x_v - (\Delta\sigma \cos\Delta\theta x_u - \Delta\sigma \sin\Delta\theta y_u)$$
$$\Delta y = y_v - (\Delta\sigma \sin\Delta\theta x_u + \Delta\sigma \cos\Delta\theta y_u) \quad (4.5)$$

또한 계산된 각각의 correspondence 의 homography 를 가지고, 이전 절에서의 식 (4.1) 과 식 (4.2) 를 통해서, correspondence 간의 similarity 를 계산할 수가 있다.

4.2.2 Constrained region

Region-constrained clustering 에서는 region 의 크기가 정확도와 computational complexity 에 영향을 미치게 된다. 좀 더 구체적으로는, region 내의 correspondence 수가 연산량과 밀접하게 연관되어 있다. 본 연구에서는 constrained region 을 설정하기 위해 segmentation 의 정보를 사용한다.

Segmentation 의 결과들은 때때로 over-segment 된 결과를 생성하게 된다. Over-segment region 에 의해서 생성된 영역들을 각각의 constrained region 으로 설정하면, 영역 내에 존재하는 correspondence 는 개수가 작을 뿐만 아니라, 영역 내에 존재하는 correspondence 들이 inlier 의 수보다 많은 outlier 로 구성되어 특성을 잘못 기술하는 경우도 발생하게 된다. 따라서, 본 연구에서는 segment region 들이 유사한 특성을 가지고 있을 때, over-segment 된 region 들을 병합하여, 가능한 많은 수의 유사한 특성을 가진 correspondence 를 candidate region 에 포함되도록 설정하는 방법을 제안한다. 본 연구에서는 segmentation 과정을 수행하기 위해서 watershed

transform 을 사용하였고 이에 기반하여, clustering 을 위한 후보 영역을 설정하였다. 특히 watershed segmentation 방법은 이미지를 edge 기반의 disjoint set 으로 구분할 수 있기 때문에, object 를 여러 개의 parts 로 나누는데, 사용할 수 있다. 일반적인 watershed 방법 및 블록 기반의 향상된 parallel watershed 방법에 대한 설명은 5 장에서 자세히 설명한다.

Segmentation region 들간의 유사성을 정의하기 위해서, 다음과 같은 region homography 를 정의하였다. Region 의 homography matrix 는 영역 내에 있는 각 correspondence homography 의 평균값으로 정의한다. 하나의 region A_i 에 m 개의 correspondence 가 존재한다고 할 때, A_i 의 area homography 는 식 (4.6) 와 같이 region 내에 평균적인 scale, rotation, translation 을 표현하는 각 matrix 의 곱으로 표현이 가능하다.

$$H_i = \mathrm{S}\left(\sum_{j=1}^{m} \Delta\sigma_j\right)\mathrm{R}\left(\sum_{j=1}^{m} \Delta\theta_j\right)\mathrm{T}\left(\sum_{j=1}^{m} \Delta x_j\,,\sum_{j=1}^{m} \Delta y_j\right) \qquad (4.6)$$

식 (4.6) 로 표현된 각 영역에 대한 region homography 를 사용하여, 인접한 두 개의 영역간의 homography 의 similarity 를 계산한다.

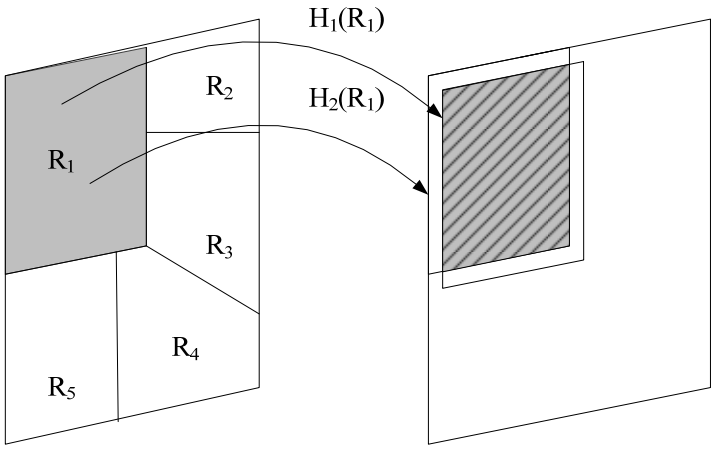


그림 4.4 Region homography projection

그림 4.4 에서 같이 인접한 두 개의 영역 R_1 과 R_2 에 대한 region homography similarity 를 생각해 보자. 식 (4.6) 에 의해서 계산된 영역 R_1 과 R_2 에 대한 homography 를 각각 H_1 와 H_2 라 할 때, 두 개의 homography 가 유사하다면, 하나의 영역을 기준으로 homography H_1 와 H_2 에 의해 projection 된 영역은 그림 4.4 과 같이 중첩될 것이다. 위와 같은 관찰을 통해서, 본 연구에서는 두 개의 homography 간 유사성을 측정하기 위한 방법으로 한 영역이 서로 다른 homography 에 의해서 projection 될 때의 overlap 된 영역의 존재 여부에 따라, homography 의 유사성 여부를 측정하였다. 이때, overlap 된 영역은 region 영역의 크기와 형태에 따라서, 영향을 받게 된다. 따라서, homography 만의 유사성을 측정하기 위한 수단으로 region 영역의 크기와 형태에 영향을 받지 않는 기준을 마련할 필요가 있다. 이를 위해서, 고정된 크기의 원 형태를(unit circle) similarity 측정을 위한 방법으로 사용하였다. 이와 유사한 접근 방법이 affine region

detector 에서 matching score 를 측정하는 수단으로도 사용되었다[20].

H_i 에 의해서 R_i 의 projected 된 영역은 식 (4.7) 과 같이 두 개의 matrix 의 곱으로 표현 가능하다. 여기서 matrix X_i^T 은 R_i 은 region R_i 내에 있는 모든 픽셀들을 나타낸다.

$$X_i^T = [x_i, y_i, 1] \;\; where\ for\ all\ (x_i, y_i) \in R_i \qquad (4.7)$$
$$X_{i'} = H_i X_i$$

위와 같은 관계를 사용해서, 각각 H_i, H_j 의 homography 를 가지는 두 개의 인접한 영역 R_i, R_j 의 region similarity 는 식 (4.8) 과 같이 표현 가능하다

$$Region\ similarity(R_i, R_j) = \begin{cases} 1, & H_i X_i \cap H_j X_i \neq \emptyset \\ 0, & H_i X_i \cap H_j X_i = \emptyset \end{cases} \qquad (4.8)$$

여기서 X_i 는 unit circle 내의 모든 픽셀들을 나타내는 행렬이다. 즉 $X_i^T = [x_i, y_i, 1]$ where for all $(x_i, y_i) \in UC$ 와 같은 관계를 가지고 있다.

위의 식에서 두 개의 projected region 이 overlap 되는 경우엔 “1” 의 region similarity 값을 가지고, overlap 되지 않는 경우엔 “0” 의 region similarity 값을 가지게 된다. Region similarity 가 “1” 인 영역들은 유사한 homography 를 가지는 영역들로 판단되어, 두 개의 영역들은 $R_1' = R_1 \cup R_2$ 의 새로운 영역으로 합쳐지며, 그 결과 clustering 과정에서 새로운 candidate region 이 생성된다.

영역별 clustering 의 정확도는 영역 내에 inlier 의 수가 outlier 의 수보다

클 때 높아진다. 제안된 방법의 candidate region 을 구성하는 방법은 over-segment region 에서 correspondence 의 수가 작은 경우에 region 병합에 의해서 정확도를 증가시켜 준다. 또한 region 별 clustering 을 수행할 때 두 개의 correspondence 들간의 similarity 를 계산하는 과정에서 correspondence 의 homography 를 사용하므로, 제안한 방법의 유사한 homography 들로 구성하는 candidate region 을 사용할 경우, 신뢰성 높은 결과를 얻을 수 있게 된다. Over-segment 된 이미지에서 모든 조합의 region 간 homography 를 조사하는 것은 많은 양의 연산을 필요로 한다. 따라서 이러한 computational complexity 를 줄이기 위해서, 인접한 영역간의 region 병합을 수행하게 된다. Segment region 에서의 인접한 영역들은 일반적으로 RAG(Region Adjacency Graph) 형태로 표현이 가능하다. 이와 같은 그래프를 사용하면, 인접한 영역간의 관계를 쉽게 알 수 있게 된다.

4.2.3 Complexity analysis

그림 4.5 는 입력 이미지를 segmentation 으로 나누고, clustering operation 을 위한 candidate region 을 구성하여, clustering 을 수행하는 과정을 보여준다. 초기의 segmentation 결과로부터 candidate region 을 형성하기 위해서, 초기 correspondence 로 계산된 region similarity 관계를 사용하며, 이를 통해 확장된 candidate region 이 생성된다. 또한, 이 candidate region 은 clustering operation 을 수행하기 위한 constraint 가 된다. 따라서, candidate region 별로 HAC 를 수행하게 된다. 최종 clustering 결과는 각각 candidate region 에서의 HAC 의 결과들의 합으로써 표현된다.

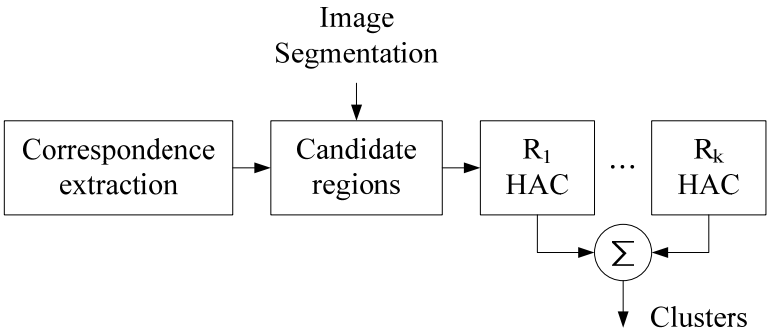


그림 4.5 The flow of the proposed clustering algorithm

N 개의 correspondence 에 대해서, 이전의 HAC 방법은 최대 N-1 개의 cluster 가 생성되며, 이 경우 HAC 알고리즘에서 설명했던 바와 같이 N-1 iteration 과정(step 2, 3, 4)이 필요하게 된다. 또한 cluster 간 similarity 를 계산하기 위해선, $O(N^2)$ 만큼의 연산이 필요로 한다. 따라서, 전체 complexity 는 $O(N^3)$ 이 된다[35]. 반면 k region 단위로 나누어진 이미지 내에서 각각 $n_1, n_2, \cdots, n_k$ 개의 correspondence 가 있다고 생각해 보면, 식 (4.9) 과 식 (4.10) 와 같은 수식이 만족된다. 따라서, 식 (4.10) 과 같은 관계에 의해서 제안한 방법은 이전 방법대비 수행 시간을 크게 줄일 수 있게 된다.

$$\bigcup_{i=1}^{k} n_i = n \,, where\ n_i \cap n_j = \emptyset \tag{4.9}$$

$$n_1^3 + n_2^3 \cdots n_k^3 \ll n^3 \tag{4.10}$$

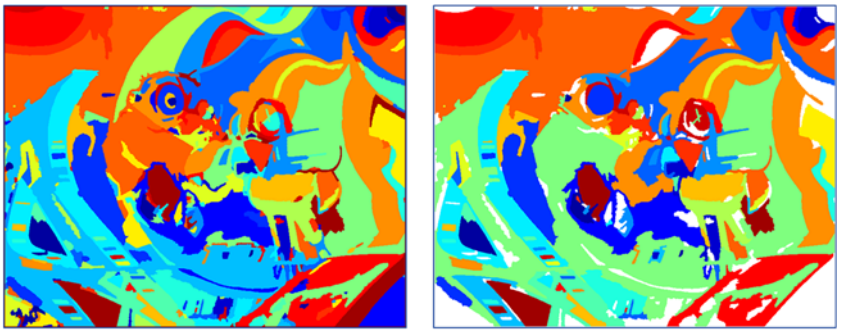
4.3 성능 평가

본 연구에서는 shared contents 를 가진 두 개의 dataset 에 대해서 실험을 진행하였다. 첫 번째 dataset 은 9 개의 model 과 23 개의 실험 이미지로 구성되어 있고[36], 또한 비교적 correpondence 의 개수가 적게 나오는 이미지들을 포함하고 있다. 두 번째 Oxford data set 은 viewpoint change, image blur, JPEG compression artifacts, illumination change 와 같은 다양한 degradation 에 의해 변형된 이미지들로 구성되어 있다. 또한 이미지의 크기가 크고, 복잡한 이미지들로 구성되어 많은 수의 correspondence 를 포함하고 있다. Oxford database 은 ground truth correspondence 를 제공하므로, 이를 통해 정량적인 matching socre 계산이 가능하다.

초기 correspondence 는 Lowe[9] 논문에서 사용한 NNDR 방법을 사용하여 생성하였다.

본 연구에서는 region 단위의 clustering 을 수행하고 있다. 그림 4.6 는 Oxford dataset 의 Graffiti 이미지에 대한 segmentation 결과와 이에 대한 candidate region 을 표시한다. 본 연구에서 사용한 watershed 에 기반한 segmentation 은 edge 를 유지하면서 object 의 part 단위별로 영역이 구분된다. 이러한 segmentation 결과에서 region 간 동일한 homography 를 유지하는 영역들을 하나의 영역으로 구성하였다. 그림 4.6 (a) 는 watershed 에 의해서 segmentation 된 영역들을 다른색들로 표현한 그림이고, 그림 4.6 (b) 는 각각의 candidate area 를 다른 색들로 표현한 그림을 보여준다. 초기의 segmentation 결과에 비해서, 인접 영역간 유사한 homography 를 가지는 영역들끼리 합쳐지는 것을 볼 수 있다. 흰색으로 표시된 영역은

correpondence 가 존재하지 않거나 하나의 correspondence 만 존재하는 영역을 보여준다. Clustering 은 correspondence 간 similarity 연산을 수행하기 위해서, 최소 2개 이상의 correspondence 를 필요로 하기 때문에, 이 영역은 clustering 연산에서 제외하게 된다.



(a) (b)

그림 4.6 Candidate areas for clustering over Graffiti image. (a) Segmentation areas (b) Candidate areas

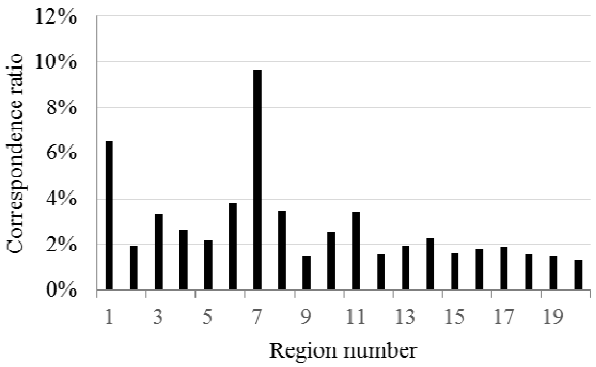


그림 4.7 전체 대비 candidate region 내의 correspondence 비율

그림 4.7 은 전체 이미지에서 발생하는 correspondence 의 개수 대비 candidate region 내에서 발생하는 correspondence 수의 비율을 보여준다. 그림은 이중 개수가 많은 상위 20 개의 영역들에서의 비율을 표시하고 있다. 모든 candidate region 들에서 correspondence 비율은 10% 보다 작다. 이와 같은 분포는 식 (4.10) 관계 따라 전체 수행시간을 크게 줄일 수 있게 된다.

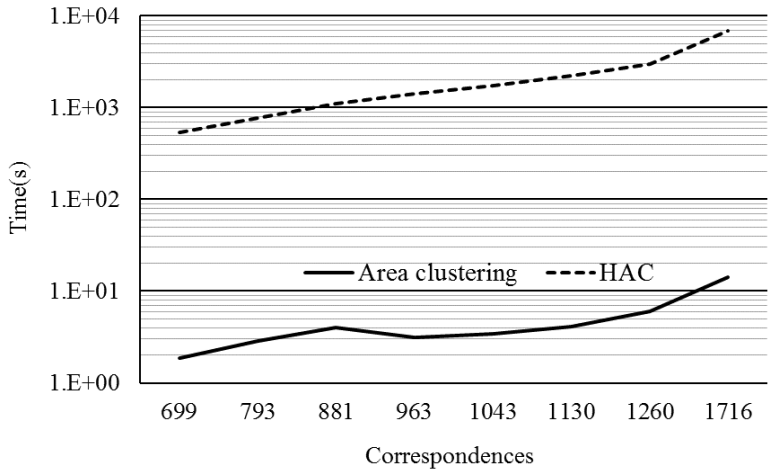


그림 4.8 Area clustering 과 HAC 의 수행시간 비교

그림 4.8 은 Oxford Graffiti dataset 을 사용해서, correspondence 수에 따른 수행시간을 보여준다. 가로축의 값은 NNDR threshold 를 달리했을 때 생성되는 correspondence 의 수를 표시하고, 세로축의 값은 log scale 로 표시된 수행시간을 보여준다. 실험은 2.67Ghz 로 동작하는 single-core Intel i5-750 processor 을 사용하였다. Correspondence 의 수가 증가함에 따라서, HAC 와 area-clustering 간의 수행시간 차이가 더 크게 증가하게 된다. 실

험에서 area-clustering 을 위한 수행시간은 segmentation 과정을 통한 candidate-region 생성시간을 포함하고 있다.

표 4.1 Recognition results over Oxford dataset

	Bikes	Graffiti	Boat	Leuven
Proposed cluster size	733	1086	917	1096
Proposed Matching score	72.7%	65%	83.2%	78.8%
HAC cluster size	886	1253	1062	1189
HAC Matching score	65.4%	58.9%	82.9%	74.9%

표 4.1 은 제안된 clustering 방법과 HAC 방법의 정확도를 보여준다. 정확도를 측정하기 위해서, feature matching 알고리즘에서 많이 사용되는 matching score 의 값을 사용하였다. 실험에서는 이미지 쌍당 수백 개의 initial feature correspondence 를 보이는 Oxford dataset 에 대해서 전체 이미지에 대한 clustering 방식을 적용했을 때의 결과와 area clustering 방식을 적용했을 때의 실험 결과를 보여준다. 실험에서 사용한 dataset 에서는 affine transform 값이 제공된다. 따라서, 정확한 matching score 값의 계산이 가능하다. 제안된 방법의 matching score 는 기존의 HAC 방법 대비 높다. 제안된 방법의 clustering size 는 각 candidate region 에서의 clustering 결과의 합으로 표현된다. 제안된 방법의 clustering size 는 HAC 의 방법의 결과

와 비교해서 줄어들지만, 높은 matching score 값을 가지고 있다. 이와 같은 결과는, 제안된 방법이 효과적으로 outlier 를 제거하는 것을 의미한다.

그림 4.9 그림 4.10 에 사용된 모델 이미지

그림 4.9 은 [36] 논문에서 사용한 dataset 에 대한 model 이미지를 보여준다. 이와 같은 model 을 사용했을 때, HAC 방법과 제안한 방법의 실험 결과는 그림 4.10 과 같다. 그림 4.10 (a), (c), (e), (g) 는 HAC 에 의한 실험 결과 이미지를 보여주며 (b), (d), (f), (h) 는 제안된 방법을 사용했을 때의 실험 결과를 보여준다. 결과에서 파란점은 각각의 방법을 사용해서, inlier 로 판단된 correspondence 에 해당하는 위치를 표시한다. Cluster 의 수는 제안된 방법으로 획득된 실험 결과에서 적게 나오지만, 제안된 방법의 경우 object 위에만 결과가 존재하는 것을 확인할 수 있다. 이는 area clustering 방법이 candidate area 의 영역별로 clustering 을 수행하기 때문에, candidate area 밖의 outlier 에 의해서 잘못 clustering 이 될 가능성이 낮다는 것을 의미한다.

그림 4.10 HAC versus area clustering (a), (c), (e) and (g) show the results by HAC. (b), (d), (f), (h) show the results by the proposed area clustering.

Recognition 의 정확도를 측정하기 위해서, recall 과 precision rate 를 사용하였다. Recall 과 precision 은 두 이미지 사이에서의 correct match 와 false match 의 수를 통해서 계산된다. Positive 나 negative 로 표현된 match 들은 4 가지의 조합, TP(True Positive), FP(False positive), TN(True Negative), FN(False Negative) 중 한 유형으로 분류된다. 이와 같은 유형들은 표 4.2 같은 matrix 형태로 표현 가능하다.

표 4.2 Confusion matrix

	Actual positive	Actual negative
Predicted positive	TP	FP
Predicted negative	FN	TN

이때, recall 과 precision 은 식 (4.11), 식 (4.12)과 같이 정의된다.

$$recall = \frac{TP}{TP + FN} \tag{4.11}$$

$$precision = \frac{TP}{(TP + FP)} \tag{4.12}$$

표 4.3 은 그림 4.10 의 precision 과 recall 값을 보여준다. 전체적으로 제안된 방법의 precision 값이 높다. 그러나 recall 의 값은 area clustering 방법이 candidate region 내의 correspondence 만을 가지고 수행하기 때문에, HAC 방법보다 낮은 값을 가진다.

표 4.3 그림 4.10 의 결과에 대한 recall 과 precision

	(a)(b)	(c)(d)	(e)(f)	(e)(h)
Proposed cluster size	51	79	340	100
Proposed recall	0.71	0.71	0.85	0.67
Proposed precision	0.96	0.95	0.99	0.95
HAC cluster size	78	124	408	120
HAC recall	0.81	0.92	0.97	0.70
HAC precision	0.72	0.78	0.95	0.83

이러한 방법은 전체 영역에 대해 clustering 을 수행한 기존 방법과는 달리 candidate region 단위로 연산이 이루어지기 때문에, clustering 을 위해 조사하는 correspondence 의 수가 region 내의 correspondence 로 제한되어, 적은량의 연산만으로도 계산이 가능하게 된다. 또한 영역 외부에 존재하는 outlier 의 영향을 받지 않기 때문에, 좀 더 정확한 결과 획득이 가능하게 된다.

제5장 Block based parallel watershed segmentation

많은 응용 분야에서 실시간 영상 분할을 필요로 하고 있으며, 여러 가지 영상 분할 알고리즘 가운데, watershed 알고리즘은 경계가 불분명한 이미지에서 효과적으로 이미지를 나눌 수 있는 있는 방법이다[40][1]. 이 방법은 이미지의 크기에 비례하여 수행 시간이 증가하기 때문에, 실시간 처리가 가능한 watershed에 대한 연구가 진행되어 왔다[41][42][43][44]. 이미지의 크기가 증가함에 따라 수행시간이 증가하는 문제점에 대한 개선 방법으로 이미지를 블록 단위로 분할하고, 병렬 연산을 수행하거나 일부 블록에 대해서만 연산을 수행하는 방법이 제안되었다. 블록 기반의 watershed는 각 블록 경계에서 영역이 분할되기 때문에 이를 통합하는 동작이 필요하다. 기존의 방법에서는 블록 기반 연산을 통한 영역 분할 결과가 프레임 기반 watershed보다 더 많은 영역으로 분할되는 문제가 있었다. 본 연구에서는 블록 경계에서 영역이 통합되거나 분리되는 경우를 관찰하여, 이 관찰을 근거로 경계 영역을 통합 방법을 제안하였다. 제안 방법은 watershed 및 label 매칭 동작이 블록별로 독립적으로 수행 가능하기 때문에 병렬 처리를 통하여 속도를 크게 향상시킬 수 있다.

5.1 Watershed transform

영상 분할은 이미지를 동질 영역들의 집합으로 나누는 기법으로 여러 영상 분석 방법들에 있어서 중요한 도구로 사용되며, 그 응용 분야 역시 매우 많다. 그 중에서 watershed 기반 영상 분할 방법은 edge 기반의 방법과 달리 영역의 경계가 희미한 경우에도 효과적으로 영역을 구분할 수 있기 때문에 가장 널리 사용되는 방법이다. 최근 여러 응용분야에서 영상 분할의 실시간 처리를 요구하는 경우가 늘어나고 있는데, 영상 분할 방법에서는 비교적 많은 연산이 수행되기 때문에 실시간 처리가 용이하지 않다. 따라서 실시간 영상 분할에 활용할 수 있는 watershed 알고리즘에 대한 연구가 이루어지고 있다.

본 연구에서는 Vincent-Soille watershed transform 을 사용하였다[39]. 이 방법은 이미지를 지형학적으로 고려하는데, 이미지의 gray level을 그 지점에서의 높이로 간주한다. Local 영역에서 가장 낮은 높이를 root라 정의하고 이 root에서 출발하여 영역을 병합해 나간다. 그림 5.1 은 이와 같이 영상을 지형학적으로 고려하는 개념을 설명하는 예를 보인다. Root와 인접한 픽셀들의 gradient가 단조 증가하는 경우 이 픽셀들은 root와 같은 영역에 있는 것으로 판단한다. 이러한 픽셀들을 기준으로 주변 픽셀들의 gradient를 조사하여 영역을 다시 확장하며, 이 영역은 동일한 root를 갖는다. 같은 root를 갖는 영역을 catchment basin (CB) 이라고 하고, 다른 root로부터 확장된 영역과 합쳐지지 않도록 하기 위해서 구분하는 경계선을 watershed line 이라고 한다. 이와 같이 영상 내에서 root들을 설정하고, 이 root들을 시작점으로 하여 전체 영상을 분할한다.

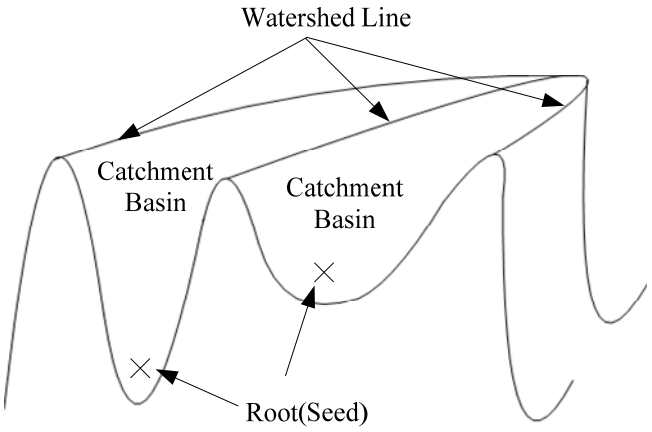


그림 5.1 Watershed 알고리즘 개념

Vincent-Soille watershed 알고리즘은 영상의 gradient 를 입력으로 받으며, 각각의 CB 이 동일한 label 을 가지는 disjoint set 결과를 출력한다. Watershed 알고리즘은 크게 두 단계의 과정으로 진행된다.

첫째, sorting 과정을 수행한다. Gradient 입력을 받으면 우선 gradient 를 기준으로 영상의 모든 픽셀들을 정렬한다. 이 과정은 가장 낮은 gradient 값(hmin) 으로부터 가장 큰 gradient 값(hmax) 값 순으로 정렬하고, 이후의 flooding 과정에서 특정 값의 gradient 에 해당하는 픽셀을 바로 접근하기 위해서 이러한 과정을 수행한다.

두 번째, flooding 과정을 수행한다. flooding 과정은 root 를 찾는 과정과, 이를 가지고 flooding 과정을 수행하여 CB 을 만드는 과정으로 나뉘어 진다. Root 는 해당 gradient 값이 주변의 다른 값보다 낮은 값을 가지거나 또는 주어진 높이 h 에서 주변의 모든 값들이 주변에 label 된 값을 가지지 않는 경우를 root 로 정의하게 된다. Flooding 과정에서는 현재 픽셀값이 이미

label 된 픽셀과 동일한 높이의 gradient 를 가지는 경우 현재 픽셀 주변의 이미 label 된 값을 부여하게 되고 이때, 같은 높이의 gradient 값을 가지는 동일한 label 들을 plateau 라고 하게 된다. 또한 현재 픽셀의 주변의 픽셀 값이 현재 픽셀보다 낮은 이미 label 된 값을 가지는 경우는 주변 픽셀의 label 값을 부여하게 된다. Flooding 과정에서 현재 픽셀의 주변의 픽셀의 label 값들이 서로 다른 경우에는 현재 픽셀을 watershed line 으로 정의한다.

5.1.1 Predictive watershed algorithm

이전 논문에서의 Predictive watershed algorithm[43] 은 Vincent-Soille 알고리즘의 속도 향상을 위해, 연속된 video frame 에서의 temporal correlation 을 사용하여 변화된 영역에서만 watershed 알고리즘을 수행한다. 변화된 영역의 계산은 블록 단위로 판단하기 때문에 변화된 영역이 블록의 집합된 형태로 나타나게 된다. 따라서 블록 경계에서 이미 계산된 영역(unchanged area) 과 새로 계산해야 하는 영역(update area)에서 경계 부분 처리 방법이 필요하게 된다.

Predictive watershed algorithm 은 Vincent-Soille 방법과 같이 update area 내에서 가장 낮은 gradient 값으로부터 주변 영역을 확장해 나가게 된다. 이때, 그림 5.2 (a) 에서 보는 것과 같이 update area 에 위치한 trilled ball 주변은 이미 계산된 Region 1 영역을 가지고 있다. 따라서 이 영역은 Region 1로 labelling 된다. 또한 update area 에 위치한 black ball 주변은 이미 계산된 Region 2 영역을 가지고 있다. 따라서 이 영역은 Region 2 영역으로 labelling 된다. 이와 같은 방법으로 그림 5.2 (b) 와 같이 3 개의 영

역으로 segment 된 region 을 생성하게 된다.

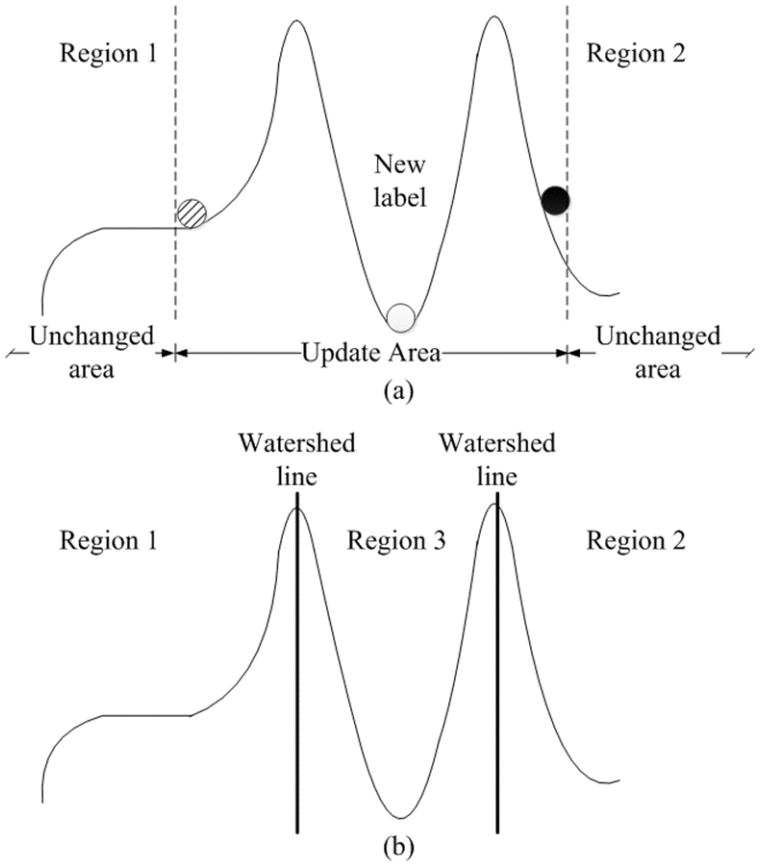


그림 5.2 Predictive watershed algorithm (a) Operation of the algorithm (b) Segmentation results

위와 같은 경우엔 변화된 부분의 블록에 대한 watershed 알고리즘이 성공적으로 수행되지만, 블록간 경계에는 다양한 경우의 수가 존재한다. 따라서 그림 5.3 (a)와 같은 gradient surface 를 갖는 영상에 대해서 predictive watershed algorithm 을 수행하면, Region 1 과 update area 의 경계에서는 주변 Region 1 의 label 이 전달되지만, update area 와 Region 3 경계에서는

update area 가 이미 가장 낮은 gradient 값으로부터 새로운 label 을 가지고 있기 때문에, 그림 5.3 (b) 와 같이 update area 와 Region 3 경계가 나누어지게 된다. 이와 같은 결과는 frame 기반의 watershed 에서는 2개의 영역이 생성되어야 하지만, 블록 기반의 연산의 결과로 over-segmentation 이 발생하는 경우가 된다.

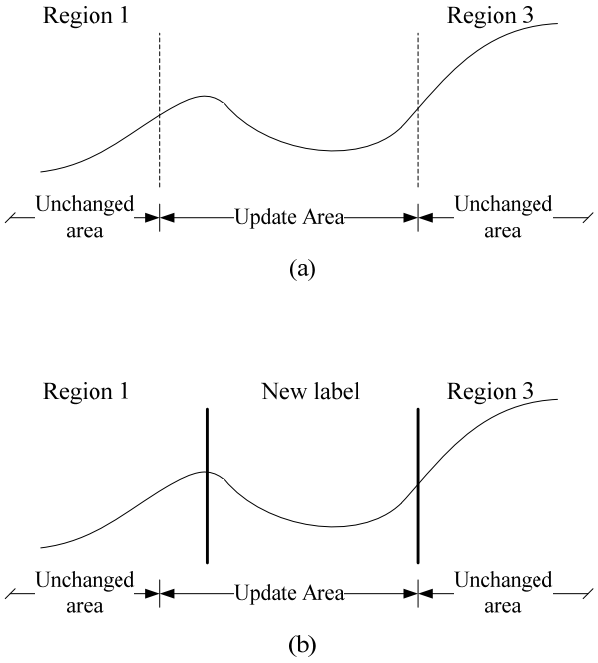


그림 5.3 Over-segmentation by predictive watershed algorithm (a) The gradient surface (b) Segmentation result

따라서 본 연구에서는 위와 같이 블록 경계에서 발생하는 over-segmentation 문제를 개선하기 위한 연구를 진행하였다.

5.2 Parallel watershed segmentation

본 절에서는 블록 기반 watershed 방법을 제안한다[41]. 그림 5.4 는 본 연구에서 사용한 segmentation 방법의 flow 를 보인다.

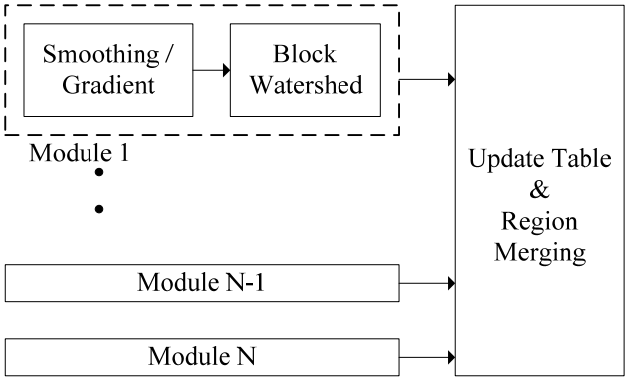


그림 5.4 Parallel watershed system

5.2.1 Preprocessing

영상이 입력되면 우선 이미지 smoothing 블록에서 입력 이미지의 노이즈를 제거한다. 이러한 동작은 노이즈에 의해 과도하게 영역이 분할되는 것을 방지한다. 입력 이미지에 대해 smoothing 을 수행한 후에는 이미지의 gradient 를 계산한다. 본 연구에서 사용된 watershed 알고리즘에서는 입력으로 영상의 gradient 를 사용한다. Gradient 는 일반적으로 object 의 경계에서 가장 큰 값을 가지며 이는 영상 분할에 유용하게 활용될 수 있다. 이와 같은 전처리를 거친 영상은 watershed transform block 으로 입력된다.

그림 5.4 에서 Image Smoothing 과정과 Gradient Calculation 과정은 픽

셀기반의 연산을 수행하므로 수행 시간이 많이 걸린다. 따라서 제안하는 블록 기반 watershed 의 효과를 극대화하기 위해서 이미지 smoothing 과 gradient calculation 의 전처리 과정도 블록 기반으로 수행하도록 하였다. 그림 5.5 는 블록 단위로 처리하는 순서를 보여 준다.

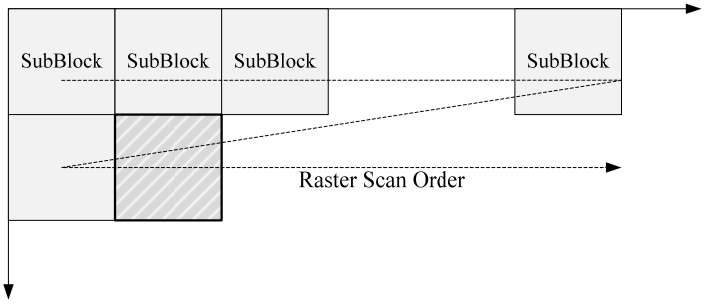


그림 5.5 Block processing order

이미지 smoothing 과정에서는 모든 element 가 '1' 인 3x3 윈도우를 가지는 mean filter 가 사용되었다. 블록 단위로 smoothing 을 수행하기 때문에 블록 경계 부분에서는 인접 블록의 픽셀 데이터가 필요하다. 그림 5.6은 블록 경계의 한 픽셀에 대한 smoothing 연산을 위해 필요한 인접 블록의 픽셀들을 보인다. 그림 5.6 에서 Block IV는 현재 처리하고 있는 블록을 의미하고, Block I, II and III 은 Block IV의 왼쪽 위, 위 및 왼쪽 블록을 나타낸다. P 로 표시된 점이 smoothing 을 하고자 하는 픽셀을 나타내고, 'X' 로 표시된 점은 P 의 연산을 위해 필요한 인접 블록의 픽셀을 의미한다. 따라서 smoothing 연산에서는 주변 블록의 1 픽셀폭에 해당하는 픽셀들의 값을 저장해야 한다.

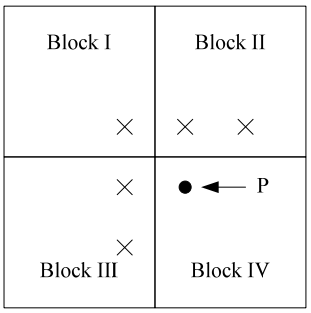


그림 5.6 Block connection

Gradient 계산에서는 morphological gradient 를 사용하였다. Morphological gradient 는 입력 이미지 f 에 대해서 morphological dilation 과 erosion 과정을 수행한다. $\oplus$ 과 $\ominus$ 을 각각 morphological dilation 과 erosion 연산으로 정의할 경우, morphological gradient 는 식 (5.1) 로 주어진다[46].

$$Gradient(f) = (f \oplus B) - (f \ominus B) \tag{5.1}$$

여기서 B 는 모든 element 가 1인 3x3 structuring element 를 나타낸다. Gradient 의 계산에서는 smoothing 과정과 마찬가지로 블록 경계에서 인접 블록의 1 픽셀폭에 해당하는 픽셀의 smoothing 이미지를 저장해야 한다. 이와 같은 블록 기반 전처리 과정을 거친 영상은 watershed transform block 의 입력으로 사용된다.

5.2.2 Block-based watershed segmentation

제안된 방법은 이미지를 블록으로 나누고 각각의 블록을 독립적으로 처리한다. 따라서, 하드웨어 구현에 적합하도록 되어 있고, 병렬 처리를 위한 확장이 가능하다. 이미지 단위로 segmentation 을 처리시에는 전체 이미지에 대한 smoothing 과정이 완료된 이후에 gradient 계산이 가능하고, 전체 이미지에 대한 gradient 가 완료된 이후에 watershed 알고리즘의 시작이 가능하다. 또한 이미지 크기에 해당하는 이전 결과들이 저장이 되어야 한다. 따라서 이에 해당하는 저장 공간이 필요하게 된다. 반면 블록 단위로 처리시에는 블록의 크기에 해당하는 저장 공간만을 가지고 처리하고 다음 블록을 처리하거나, 또는 동시에 동일한 모듈을 여러 개 두어 동시에 처리가 가능하기 때문에, distributed memory system 에도 적용이 가능하게 된다.

하지만 블록 단위로 watershed 를 수행했기 때문에 같은 영역이 여러 블록에 걸쳐 있을 때 블록 경계에서 영역이 분할되며, 그에 따라 이미지 단위의 watershed 보다 매우 많은 영역으로 나누어진다. 이미지 기반 watershed 와 동일한 수의 영역으로 분할하기 위해서는 블록의 경계에서 분할된 영역이 실제 object 의 경계인지 혹은 블록 단위 연산에 기인한 것인지를 조사해야 한다. 만약 블록 경계에서 인접한 두 영역이 실제로는 하나의 영역인 경우에는 두 영역의 label 을 같은 값이 되도록 해야 한다.

그림 5.7 는 하나의 object가 2개의 블록 (Block I, Block II) 에 나뉘어 위치하고, 그에 따라 블록의 경계에서 2개의 영역으로 분할된 예를 보인다.

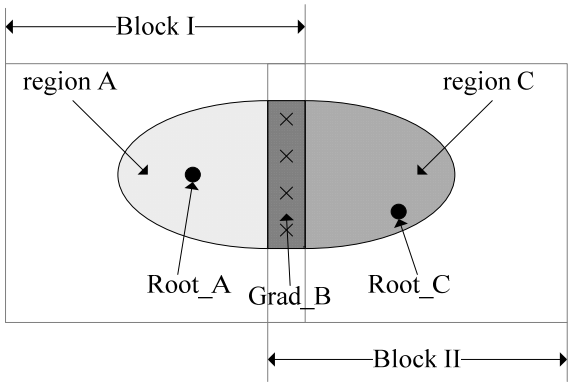


그림 5.7 Region across the block boundary

제안된 방법에서는 블록 단위의 연산을 위해 각 블록별로 Vincent-Soille watershed 알고리즘을 수행하고 그 결과를 각 블록마다 독립적으로 저장한다. Block I 에서 watershed 를 수행하면, region A 가 생성되는데 이 region A의 root의 gradient 값을 Root_A 라 한다. 그 다음 block II 에 대한 watershed 를 수행하는데, 이 때 이전 블록의 1 픽셀폭만큼의 픽셀들을 포함하여 watershed를 수행한다. 그 결과로 region C 가 분할되면 이 영역의 root 값을 Root_C 로 표시한다. 두 영역 경계에서는 1 픽셀폭만큼의 중첩된 영역이 발생하는데, 이 영역을 boundary area 라고 나타낸다. 그림 5.7 에서는 boundary 영역에서의 gradient 값을 Grad_B 로 표시하였다. 이 boundary area 에서는 두 개 sub-block 에서 수행한 watershed 결과를 동시에 가지게 된다.

이전의 연구에서는 블록 경계를 처리하기 위해 블록 경계에서의 추가 1 픽셀의 정보만을 사용할 경우, 이전 절에서 살펴본 바와 같이 over-segment 결과를 생성하게 된다.

본 연구에서는 블록의 경계에서 merge/split 판단을 하기 위해서, region 의 root 정보를 함께 사용하였다. Region 의 root 정보를 사용하는 것은 이를 저장하기 위한 추가적인 공간이 필요하고, merge/split 판단을 위한 추가적인 연산이 필요하지만, root 의 정보를 사용함으로써, 정확한 merge/split 판단이 가능하게 된다. Watershed 알고리즘에서 root 의 수는 label 의 수와 일치한다. 그림 5.8 은 Vincent-Soille watershed 알고리즘의 pseudo-code 를 설명하고 있다. Watershed 알고리즘은 입력으로 gradient 이미지를 사용하며, output 으로 픽셀 단위의 label 을 부여하는 동작을 수행한다. 이를 위해 영상내에 gradient 값을 sorting 하여, 낮은 gradient 값으로부터 순차적으로 주변 영역을 비교하는 과정을 수행하게 된다. 이때, local minimum 에는 새로운 label 을 부여하게 되는데, 이때의 local minimum 의 값이 root 값이 되게 된다. 따라서, 그림 5.8 에서와 같이 영역 label 에 대한 root 값은 watershed 연산 과정에서 저장이 가능하다.

Vincent-Soille watershed algorithm

```
1   Input    : Gradient Image
2   Output : labelled watershed image
3   /* SORT pixels of gradient values */
4   /* Start flooding */
5   for h=h_min to h_max do
6       output [P] = MASK;
7       if (output [ neighbors of P] !=0) then
8           enqueue (P);
9           distance (P) =1;
10      end
11      while (queue is not empty) then
12          /* Label P is a neighbor of other CB*/
13          /* Label P is plateau pixel */
14      end
15      if (output [P] is a MASK) then
16          output [P] = new label;
17          Root [new label] = h;
18      end
19  end
```

그림 5.8 Root information in Vincent-Soille watershed algorithm

Root 의 값은 그림 5.8 과 같이 표현된 Vincent-Soille 알고리즘에서 새로운 label 을 부여하는 과정에서 저장이 가능하다. 이와 같이 저장된 값을 가지고, 그림 5.7 과 같은 경계에서 root 의 gradient 값과 boundary 에서의 gradient 값을 가지고 발생하는 모든 경우에 대해 조사하게 된다. 그림 5.9 와 같은 경우의 수가 발생한다.

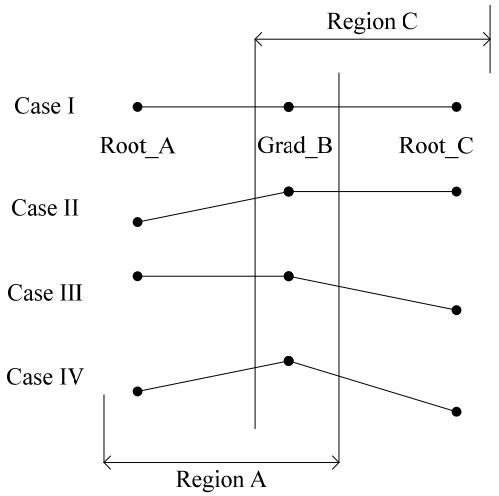


그림 5.9 Four possible relationship among Root_A, Grad_B and Root_C

Case I 은 Root_A, Grad_B, Root_C 의 값이 모두 같은 경우를 나타낸다. Case II 는 동일한 값을 가지는 Grad_B 와 Root_C 보다 Root_A 값이 작은 경우를 나타낸다. Case III 는 Root_A 와 Grad_B 가 동일하고 이 값이 Root_C 보다 큰 경우를 나타낸다. Case IV 는 Grad_B 의 값이 Root_A 와 Root_C 보다 큰 경우를 나타낸다. 이때, Root_A 와 Root_C 간의 크기는 관계가 없다. 왜냐하면 이것은 Root_A 와 Root_B 는 watershed 알고리즘의 특성상 식(5.2) 와 같이 항상 Grad_B 보다 작거나 같기 때문이다.

$$\begin{aligned} Root\_A &\leq Grad\_B \\ Root\_C &\leq Grad\_B \end{aligned} \tag{5.2}$$

그림 5.9 에서 보이는 4 가지 경우는 경계면에서 발생하는 모든 경우를 나타낸다. 따라서, 이와 같은 경우들에 대해 merge/split 을 정확히 수행하면

over-segment 문제 해결이 가능하다. 표 5.1 은 위와 같은 경우에 대해서, 판단하는 방법을 정리해서 보여준다. Case I, II, III 는 다른 블록내에 있는 두 개의 영역이 합쳐져야 하는 경우를 나타낸다. Case IV 는 두 개의 영역이 분리되어야 하는 경우를 보여준다. 이때, Case II, III 는 각각 두 개의 영역을 하나로 합치면서, root 의 값은 해당 영역의 가장 작은 값 Root_A, Root_C 로 update 가 필요하게 된다. 그림 5.10 은 이와 같은 블록간 region merge/split 을 수행하기 위한 pseudo-code 를 나타낸다. 이 과정은 입력으로 블록간 segmentation 된 결과를 사용하고, output 으로 병합되어야 하는 label 간 lookup table 을 생성하게 된다.

표 5.1 Decision for region merge/split

Case	Condition	Decision	Updated root
I	Grad_B = Root_C Grad_B = Root_A	Merge	Root_A or Root_C
II	Grad_B = Root_C Grad_B > Root_A	Merge	Root_A
III	Grad_B > Root_C Grad_B = Root_A	Merge	Root_C
IV	Grad_B > Root_C Grad_B > Root_A	Split	-

Algorithm 2 : Region Merging Criteria

Input : For Reference / Current Block
Root of region / Region Label
Boundary Gradient
Output : Region Lookup Table

```
1   If (root (C) == Gradient(B)) then
2   |  If (Gradient(B) == root (A)) then
3   |     Merge (A,C) and update table;
4   |  Else if (Boundary Gradient > root(A)) then
5   |     Merge(A,C);
6   |     root (C) = root (A);
7   |  end
8   end
9   If (root (C) < Boundary Gradient) then
10  |  If (root(A) == Boundary Gradient) then
11  |     Merge (A,B) and update table;
12  |     root(A) = root(C);
13  |  Else
14  |     Boundary pixel is WSHEDline;
15  |  end
16  end
```

그림 5.10 Region merging algorithm flow

5.2.3 Block-based skip of flooding operations

본 연구에서 제안한 방법은 watershed 연산을 독립적인 블록 단위로 처리한다. 이러한 블록 단위로 영상을 분할하는 경우 일부 블록 내에서는 영상이 분할되지 않고 블록 전체가 하나의 영역이 될 수 있다. 이러한 블록에서는 watershed 연산 과정 중에서 flooding 연산을 skip 하여 연산량을 줄일

수 있다. 이는 연산량을 효과적으로 감소시킬 수 있는 방법이 된다. 이와 같이 블록이 하나의 segment 만으로 구성되는 경우는 블록내에 root 의 값이 1개이거나 블록내의 모든 gradient 값이 동일한 경우이다. 이와 같은 판단은 블록내의 gradient 값을 sorting 하는 단계에서 가능하다. 그림 5.11 은 flooding 연산 skip 을 위한 flowchart 를 보여준다.

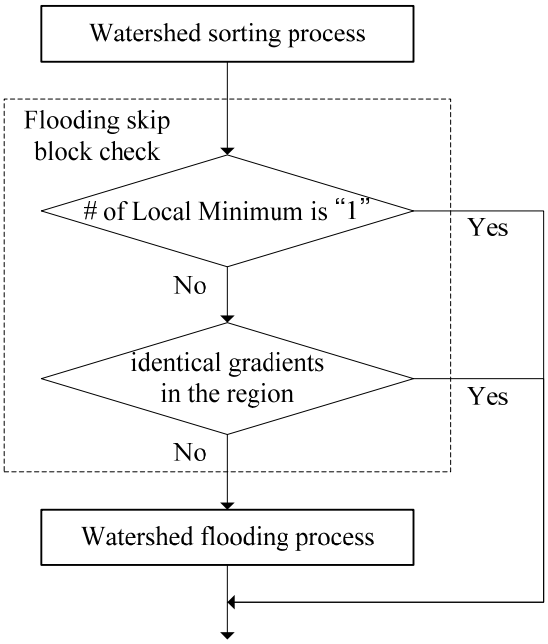


그림 5.11 Decision for skipping the flooding operation in a block

5.2.4 Update table and region merging

이 단계에서는 region lookup table 을 이용하여 label 을 정리하는 과정을 수행한다. Region lookup table 은 2 개의 column으로 구성되며 서로 merge

되어야 하는 영역들 간의 관계를 표현하고 있다. 두 개의 영역간 관계를 표현하는 table 에서 낮은 값의 label 을 첫 번째 column 에 위치시키고, 여러 단계의 연결 관계에서 경로압축(path compression) 방법으로 최종적으로 사용할 label 을 가르치도록 table 을 update 하게 된다. 그림 5.12 는 이와 같은 과정의 예를 보여준다. 그림 5.12 (a) 는 초기 영역간 관계를 저장하고 있다. 이와 같은 영역 관계에서 label 2 는 label 1 와 merge 가 되어야 하는 영역을 표시하고, label 3 는 label 2 와 merge 되어야 하는 영역을 표시한다. 따라서 label 1,2,3 은 하나의 label 을 가져야 한다. 최종적으로 정리된 lookup table 은 그림 5.12 (b) 와 같게 된다.

No	1st column	2nd column
①	1	2
②	2	3
③	1	4
④	6	7
⑤	1	6

(a)

No	1st column	2nd column
①	1	2
②	1	3
③	1	4
④	1	7
⑤	1	6

(b)

그림 5.12 Example of arranging region label (a) Initial lookup table (b) Arranged lookup table

정리된 region lookup table 을 통해 서로 병합되어야 하는 영역의 label update 를 수행하게 된다.

5.3 성능 평가

이전의 Vincent-Soille watershed 알고리즘은 전체 영상의 global 정보를 사용하기 때문에, 블록 단위로 나누어서 병렬처리하는데, 어려움이 있다. 본 연구에서 제안한 블록 기반의 접근 방법은 하드웨어 구현에 적합하도록 각각의 블록이 독립적으로 watershed 를 수행하도록 되어 있다. 또한 블록 단위로 연산이 이루어지기 때문에, 하나의 segment 로만 구성되는 블록이 발생하게 되고, 이를 사전에 예측하여, watershed 에서의 flooding 과정을 수행하지 않음으로써, 연산량 감소가 가능하다.

제안한 방법의 watershed 결과는 Vincent-Soille 알고리즘을 사용한 watershed 의 결과와 동일한 region 개수를 가진다. 그림 5.14 은 4 개의 영상에 대해서 Vincent-Soille watershed segmentation 방법과, 제안된 방법을 적용하였을 때의 block based segmentation 결과를 보여준다. 결과에서는 영역의 분할된 위치가 다르게 분할된 부분이 있는데, 그림 5.13 은 이 차이를 설명하는 예를 보인다[45]. 그림 5.13 에서 sub-block I 에 존재하는 root 의 위치를 흰색공으로 나타내고, sub-block II 에 존재하는 root의 위치를 검은색공으로 표시한다. 두 블록 경계는 동일한 gradient 를 갖는 plateau 위에 있다. Vincent-Soille 알고리즘을 수행할 때에 데이터의 scan 순서, 즉 sorting 후에 FIFO 에 저장된 데이터의 순서는 좌측에서 우측으로 처리되는 것으로 가정한다. Vincent-Soille 알고리즘의 flooding 과정을 수행하면 흰공으로부터 시작된 영역이 ① 의 위치에 도달하게 되고, 검은색공으로부터 시작된 영역은 ② 의 위치까지 flooding 과정을 수행하게 된다. Plateau area 에서는 동일한 gradient 값을 가지므로 FIFO 에 저장된 순서인

① 의 경계에 있는 픽셀들부터 주변영역과 비교하며 영역을 확장해 나가게 된다. 따라서 흰색공으로부터 출발한 영역으로 통합되게 되고, ② 의 경계까지 영역을 확장하게 된다. 따라서 최종적으로 ② 의 경계에서 Vincent-Soille 알고리즘의 watershed line 이 발생하게 된다. 제안한 방법에서는 case IV에 해당하기 때문에 영역이 분할되는데, 블록의 경계에서 watershed line 이 발생한다. 만약 ③ 의 위치에 블록의 경계가 존재하게 되면, ③ 의 위치에서 watershed line 이 발생하게 된다. 따라서 Vincent-Soille 알고리즘 기반 watershed 와 제안한 블록 기반 watershed 의 결과에 차이가 발생한다. 그러나 동일한 gradient 값을 갖는 plateau 은 영상내에서 blurring 이 심하거나, 영역분할을 위한 경계가 명확하지 않은 부분이다. 따라서 이러한 차이는 큰 문제를 발생시키지 않는다.

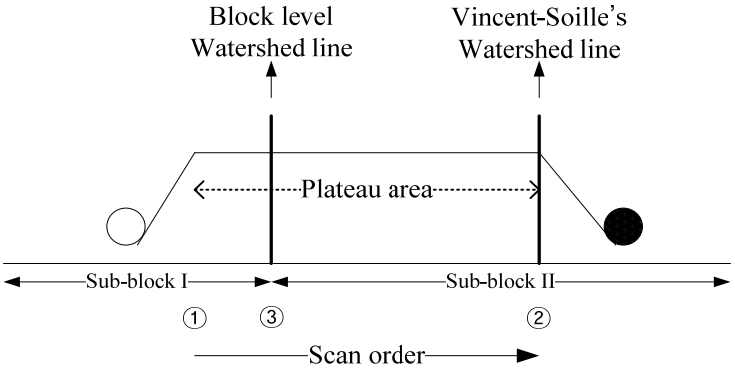


그림 5.13 Vincent-Soille 방법과 제안된 방법의 경계부분 차이

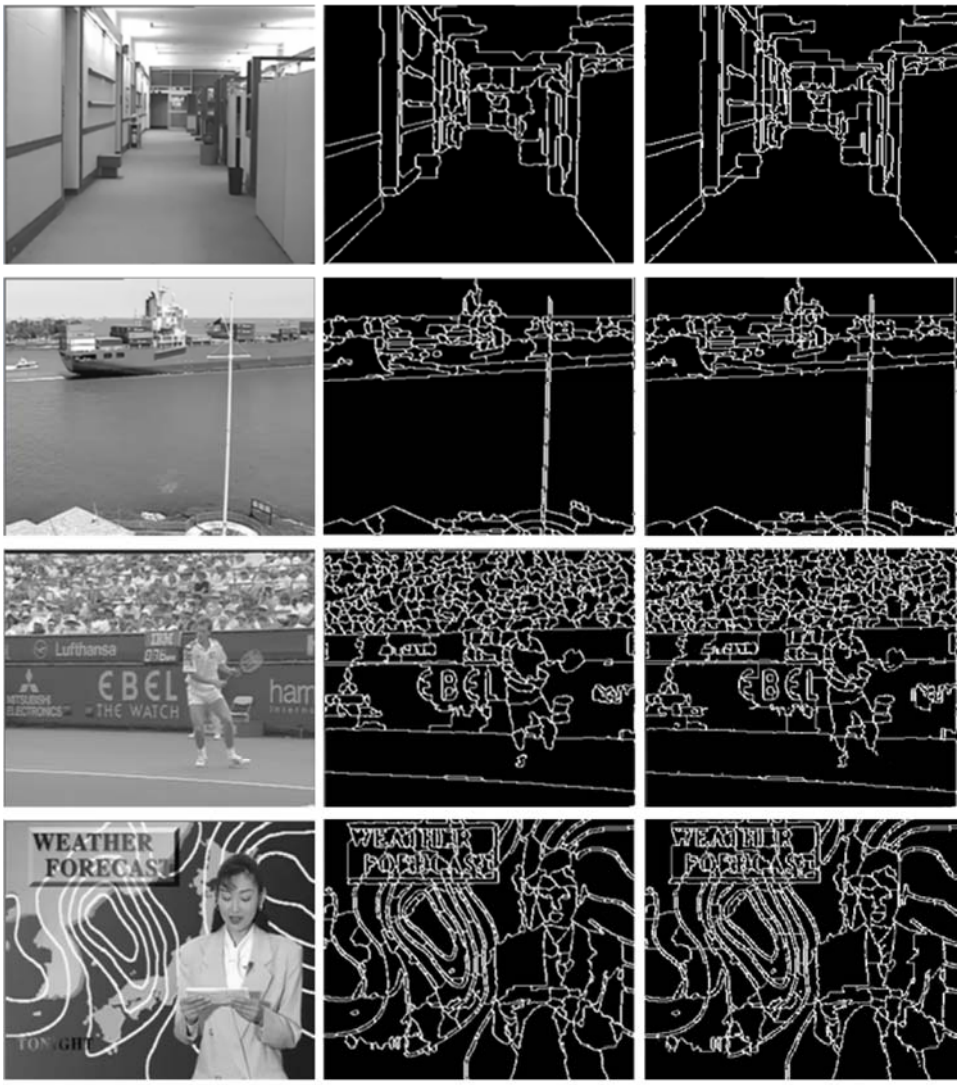


그림 5.14 Segmentation result (a) original image (b) Segmentation by Vincent-Soille algorithm (c) Segmentation by proposed algorithm

그림 5.15 는 제안한 방법을 프로세서의 수를 달리해서, 수행하였을 때의 결과를 보여준다. 각각의 processor 에서 계산된 결과는 boundary 를 계산하기 위한 연산과정을 거치기 때문에 최종적인 수행시간은 두 개의 processor 를 사용했을 경우 하나의 processor 를 사용하는 경우 대비 1.9 배의 속도 향상이 발생하고, 네 개의 processor 를 사용했을 경우 3배 정도의 속도가 향상된다.

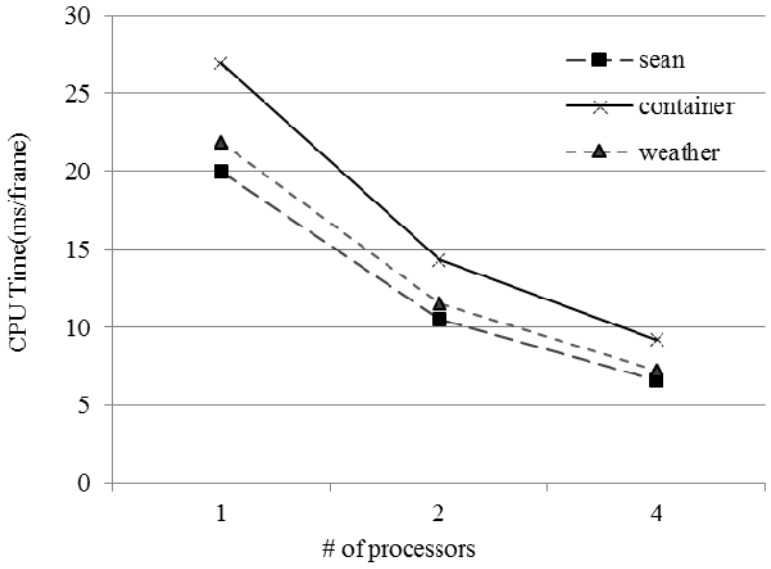


그림 5.15 Computation time versus the number of processors

표 5.2 는 16x16 의 블록 단위로 제안된 방법을 수행했을 때 flooding 알고리즘을 skip 하는 블록의 비율을 나타낸다. 이는 단일 영역으로 segment 되는 영역이 많이 분포되어 있는 영상에서 높게 나타난다.

표 5.2 Flooding operation skip 발생 비율

	Container	Foreman	Hall monitor	Sean	Weather
Flooding skip block	21%	6%	10%	7%	10%

표 5.3 은 전체 watershed 알고리즘의 전체 수행 시간 대비 flooding operation 의 수행 시간의 비율을 보여준다. Flooding 연산은 하나의 픽셀에 대해서, 주변의 여러 픽셀들을 조사하는 과정을 필요로 하기 때문에, sorting 과정에 비해서 많은 연산을 필요로 하는 부분이다. 실험적으로 5 개의 sequence 에 대해서 약 90% 정도의 수행 시간을 flooding 과정에서 필요로 한다. 따라서, 제안된 flooding skip 방법은 전체 연산량을 줄일 수 있는 효과적인 방법이 된다.

표 5.3 Execution time of flooding skip effect

	Container	Foreman	Hall monitor	Sean	Weather
Flooding operation ratio	89.5%	90.5%	89.9%	90.5%	92%

이전의 predictive watershed 알고리즘은 전체 영상을 블록으로 나누지 않고, 영상내에서 변화된 부분을 찾고 변화된 부분을 하나의 단위로 watershed 를 수행하는 방법이다. 따라서, predictive watershed 와의 비교를 수행하기 위해서, 연속된 프레임에서 변화된 부분에 대해 predictive

watershed 와 제안된 방법을 사용한 결과를 비교하였다. Predictive watershed 에서는 변화된 영역의 연결된 블록들을 하나의 단위로 연산하는 방법을 사용하였다. 그림 5.16 (a) 은 Hall monitor 영상의 20번째 프레임을 보여주고, (b) 는 해당 프레임에 대한 Vincent-Soille 알고리즘의 결과이다. 그림 5.16 (c) 는 이전 프레임에서 변화된 부분을 회색 부분으로 나타낸 것으로 회색 부분은 새롭게 watershed 를 수행해야 할 영역을 의미한다. 이와 같은 영역에 대해서 그림 5.16 (d) 는 predictive watershed 의 결과이고, (e) 는 제안한 방법의 결과를 보여준다. 제안한 방법에서 보다 정확한 segmentation 결과를 보임을 확인할 수 있다.

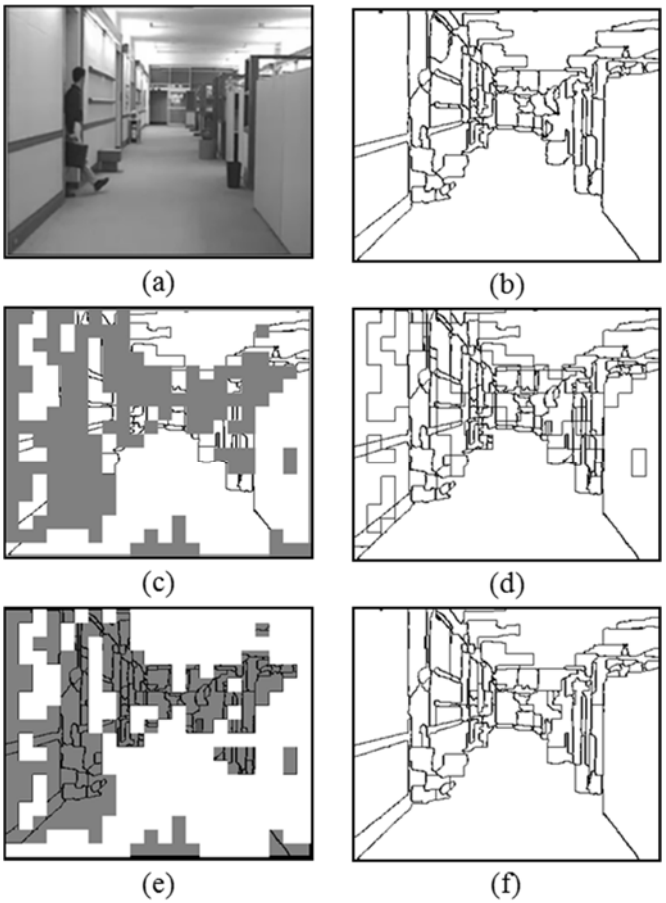


그림 5.16 Watershed segmentation results; (a) Hall Monitor sequence, (b) segmentation by Vincent-Soille algorithm, (c) updating area, (d) segmentation by predictive algorithm, (e) segmentation inside the updating region, (f) segmentation by the proposed algorithm.

제6장 결론

본 연구에서는 영역 분할 기반의 고속의 feature matching 방법을 통해 효과적으로 물체를 인식하기 위한 연구를 진행하였다. Feature matching 을 통한 물체 인식 방법은 여러 단계의 과정을 수행한다. 본 연구에서는 각각의 단계별로 성능을 향상시키기 위한 방법을 분석하고, 개선 방법들을 제안하였다.

첫 번째로, SIFT 하드웨어 에서의 속도 향상 방법으로 adaptive SIFT keypoint 생성 방법을 제안하였다. 기존에 keypoint 를 조절하는 방법들은 발생된 keypoint 를 post-processing 과정을 통해 sampling 하는 방법을 사용하였다. 그러나 keypoint detector 와 descriptor generator 가 병렬화된 hardware 에서는 병렬 구조를 유지할 수 없기 때문에, 이와 같은 방법을 적용하기가 어렵다. 또한 SIFT 의 contrast threshold 를 조절하여, keypoint 의 수를 줄이는 것은 contrast 가 높은 영역에만 keypoint 가 존재하여, 일부 영역에서는 특징을 기술할 특징점 자체가 없어지는 문제가 있다.

본 연구에서는 제안하는 방법은 블록 단위로 영상의 복잡도를 예측하여, 블록의 특성에 따라서 keypoint 가 상대적으로 많이 나오는 영역의 keypoint 는 조절하고, keypoint 가 적게 나오는 영역에서는 keypoint 가 유지되도록 하는 방법을 사용하였다. 이에 따라서, 전체적인 keypoint 의 분포를 고르게 유지시킬 수 있다. 이와 같은 keypoint 발생갯수는 application 에 따라 조절이 가능하다. 블록 내의 영상의 복잡도 예측은 FAST corner

detector 를 사용하였다. 이를 위해 영상의 복잡도를 예측하기 위한 hardware 를 설계하고, SIFT hardware 와 연동하여 동작하도록 하였다.

이 하드웨어는 SIFT 의 hardware 대비 추가적으로 5% 정도의 logic 을 필요로 하며, 전체 SIFT 의 Gaussian filter bank 연산 과정에 비해 빠른 처리가 가능하기 때문에, 초기의 latency 이후에는 SIFT 의 동작과 병렬 처리가 가능하다. 또한 이와 같은 방법에 의한 keypoint generation 방법은 전체 영상을 단일 contrast threshold 로 keypoint 를 조절하는 방법에 비해, keypoint 가 고르게 분포되어 있으며, matching score 측면에서도 이 방법을 적용하기 전과 동등 수준의 matching score 를 유지하고 있음을 확인할 수 있었다.

두 번째로, region-constrained feature matching 방법을 제안하였다. Local patch descriptor 의 유사성만을 이용하여, 대응 관계를 계산하면, 부분적으로 유사한 다른 부분의 descriptor 로 correspondence가 이루어질 수 있다. 따라서 이러한 초기의 correspondence 로부터 inlier 와 outlier 를 구분하기 위한 방법으로 HAC(Hierarchical agglomerative clustering)기반의 matching 을 수행하였다. 그러나 이 방법은 전체 영상에 대한 모든 correspondence 에 대해 clustering 을 수행하므로, 많은 수의 연산량을 요구한다. 본 연구에서는 이를 개선하기 위해, clustering 을 수행하기 위한 region 영역을 설정하고, 설정된 region 내에서 clustering 을 수행하였다. 초기의 segment 된 영역에서 region 간 homography 가 유사한 영역들을 합치는 방법으로 후보 영역을 구성하였고, 이를 통해 clustering 의 정확도를 향상할 수 있었다. 따라서 제안한 방법은 기존 HAC 방법에 비해 높은 matching score 를 가지

며, 1000 개의 correspondence 를 가지는 이미지에서 500 배까지 수행 시간을 단축을 확인할 수 있었다.

세 번째로, 블록 기반의 parallel watershed segmentation 방법을 제안하였다. Watershed 알고리즘은 전체 영상의 값에 대하여 순차적으로 연산을 수행하기 때문에, 병렬화가 어렵다. 또한 영상의 크기가 커짐에 따라 watershed 의 수행 속도가 급격히 저하되므로 실시간 처리가 매우 어렵게 된다. 따라서, 영상을 블록 단위로 나누어 처리하는 방법들이 연구되었는데, 이전 방법에서는 Vincent-Soille 알고리즘의 결과와는 달리 블록간 경계 부분에서 영상이 과분할되는 문제점을 가지고 있다.

본 연구에서는 watershed 수행 단위를 일정 크기의 블록으로 나누고, 블록간 독립적인 연산이 가능한 방법을 제안하였다. 블록간에 발생하는 dependency 는 블록 경계면에서의 gradient 값과 블록간 인접하고 있는 segment 의 root 값의 관계를 통해 예측이 가능하다. 블록간에 발생할 수 있는 모든 경우의 수를 처리함으로써 Vincent-Soille watershed 의 결과와 동일한 개수의 segment 가 생성 가능하다. 또한 블록 단위로 연산을 하기 때문에, 단일 segment 로 예측되는 블록에 대해서, Vincent-Soille 알고리즘의 연산 중 90% 이상을 차지하는 flooding 연산 부분을 수행하지 않도록 하는 것이 가능해졌다. 위의 방법을 multi-core system 에 적용하면, 2개의 processor 를 가지는 경우 약 1.9 배의 속도 향상이 가능하고, 4개의 processor 를 가지는 경우 약 3 배 정도의 속도 향상이 가능하다.

참고 문헌

[1] A.K. Jain, *Fundamentals of Digital Image Processing*, Prentice-Hall, 1989.

[2] M. Treiber, *An Introduction to Object Recognition: Selected Algorithms for a Wide Variety of Applications*, Springer, pp. 1-10, 145-147, 2010.

[3] K. Mikolajczyk, B. Leibe and B. Schiele, “Local Features for Object Class Recognition,” IEEE 10th International Conference on ICCV 2005, pp. 1792-1799, Oct. 2005.

[4] J.N. Wilson and G.X. Ritter, Handbook of Computer Vision Algorithms in Image Algebra, CRC Press, pp. 58-59, 2000.

[5] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond and D. Schmalstieg, "Pose Tracking from Natural Features on Mobile Phones," in Proc. of 7th IEEE/ACM ISMAR 2008, pp. 125-134, Sep. 2008.

[6] H.I. Koo and N.I. Cho, "Feature-based Image Registration Algorithm for Image Stitching Applications on Mobile Devices," IEEE Trans. on Consumer Electronics, vol. 57, no. 3, pp. 1303-1310, Aug. 2011.

[7] T. Tran and E. Marchand, “Real-Time Keypoints Matching: Application to Visual Servoing,” 2007 IEEE International Conference on Robotics and Automation, pp. 10-14, Apr. 2007.

[8] S. Battiato, G. Gallo, G. Puglisi and S. Scellato, "SIFT Features Tracking for Video Stabilization," 14th International Conference on Image Analysis and Processing 2007, pp. 10-14, Sep. 2007.

[9] D.G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” International Journal of Computer Vision, vol. 60, no. 2, pp. 91-110, Nov. 2004.

[10] T. Lindeberg, “Scale-Space for Discrete Signals,” IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 12, no. 3, pp. 234-254, Apr. 1990.

[11] M. Fischler and R. Bolles, “Random Sample Consensus: A Paradigm for Model Fitting with Application to Image Analysis and Automated Cartography,” Magazine on Communications of the ACM, vol. 24, no. 6, Jun. 1981.

[12] T. Tuytelaars and K. Mikolajczyk, "Local Invariant Feature Detectors: A Survey," Journal on Foundations and Trends® in Computer Graphics and Vision, vol. 3, no. 3, pp. 177-280, Jan. 2008.

[13] L. Juan and O. Gwun, "A Comparison of SIFT, PCA-SIFT and SURF," International Journal of Image Processing, vol. 3, no. 4, pp. 143-152, 2009.

[14] K. Yan, and R. Sukthankar, “PCA-SIFT: A More Distinctive Representation for Local Image Descriptors,” in Proc. of IEEE CVPR, pp. 506-513, Jul. 2004.

[15] H. Bay, A. Ess, T. Tuytelaars and L.V. Gool, "Speeded-Up Robust Features (SURF)," Elsevier Journal on Computer Vision and Image Understanding, vol. 110, no. 3, pp. 346-359, Jun. 2008.

[16] E.S. Kim and H.J. Lee, "A Novel Hardware Design for SIFT Generation with Reduced Memory Requirement," Journal of Semiconductor Technology and Science, vol. 13, no. 2, pp. 157-169, Apr. 2013.

[17] V. Bonato, E. Marques and G. A. Constantinides, “A Parallel Hardware Architecture for Scale and Rotation Invariant Feature Detection,” IEEE Trans. on Circuits and Systems for Video Technolory, vol. 18, no. 12, pp. 1703-1712, Dec. 2008.

[18] H.D. Chati, F. Muhlbauer, T. Braun and C. Bobda, “Hardware/Software co-design of a key point detector on FPGA,” in Proc. of IEEE Symposium FCCM, pp. 355-356, Apr. 2007.

[19] G. Michael, G. Helmut and B. Horst,“Fast Approximated SIFT,” in Proc. of ACCV 2006, pp. 918-927, Jan. 2006.

[20] K. Mikolajczyk and C. Schmid, ”A Performance Evaluation of Local Descriptors,” IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 27, no. 10, pp. 1615–1630, Oct. 2005.

[21] F.C. Huang, S.Y. Huang, J.W. Ker and Y.C. Chen, "High-Performance SIFT Hardware Accelerator for Real-Time Image Feature Extraction," IEEE Trans. on Circuits and Systems for Video Technology, vol. 22, no. 3, pp. 340-351, Mar. 2012.

[22] E. Rosten, R. Porter and T. Drummond, "Faster and better: A Machine Learning Approach to Corner Detection," IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 32, no. 1, pp. 105-119, Jan. 2010.

[23] A. Behrens and H. Rollinger, "Analysis of Feature Point Distributions for Fast Image Mosaicking Algorithms," Acta Polytechnica Journal of Advanced Engineering, vol. 50, no. 4, 2010.

[24] S. Gauglitz, L. Foschini, M. Turk and T. Höllerer, "Efficiently Selecting Spatially Distributed Keypoints for Visual Tracking," 18th IEEE International Conference on Image Processing 2011, pp. 11-14, Sep. 2011.

[25] M. Brown, R. Szeliski and S. Winder, "Multi-Image Matching Using Multi-Scale Oriented Patches," IEEE Computer Society Conference on CVPR 2005, pp. 510-517, Jun. 2005.

[26] G.P. Nguyen and H.J. Andersen, "Context-Based Adaptive Filtering of Interest Points in Image Retrieval," 9th International Conference on ISDA 2009, pp. 529-534, Dec. 2009.

[27] E. Ardizzone, A. Bruno and G. Mazzola, "Visual Saliency by Keypoints Distribution Analysis," Springer on Image Analysis and Processing–ICIAP 2011, pp. 691-699, Sep. 2011.

[28] Z. Cheng, D. Devarajan and R.J. Radke, "Determining Vision Graphs for Distributed Camera Networks Using Feature Digests," EURASIP Journal on Applied Signal Processing, vol. 2007, no. 1, Jan. 2007.

[29] V. Nannen and G. Oliver, "Grid-Based Spatial Keypoint Selection for Real Time Visual Odometry," in Proc. 2nd International Conference on Pattern Recognition Applications and Methods 2013, pp. 586-589, Feb. 2013.

[30] C. Gomila and F. Meyer, “Graph-based Object Tracking,” in Proc. of International Conference on ICIP 2003, pp. 14-17, Sep. 2003.

[31] L. Torresani, V. Kolmogorov and C. Rother, “Feature Correspondence Via Graph Matching: Models and Global Optimization,” in Proc. of Computer Vision–ECCV 2008, pp. 596-609, Oct. 2008.

[32] O. Duchenne, F. Bach, I. Kweon and J. Ponce, “A Tensor-Based Algorithm for High-Order Graph Matching,” IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 33, no. 12, pp. 2383-2395, Dec. 2011.

[33] M. Cho, J. Lee and K.M. Lee, “Feature Correspondence and Deformable Object Matching via Agglomerative Correspondence Clustering,” IEEE 12th International Conference on Computer Vision 2009, pp. 1280-1287, Oct. 2009.

[34] J.W. Jang and H.J. Lee, “Region-Constrained Feature Matching with Hierarchical Agglomerative Clustering,” VISAPP, 2014. (to be published)

[35] Xu Rui and D. Wunsch, "Survey of Clustering Algorithms," IEEE Trans. on Neural Networks, vol. 16, no. 3, pp. 645-678, May 2005.

[36] V. Ferrari, T. Tuytelaars and L.V. Gool, “Simultaneous Object Recognition and Segmentation from Single or Multiple Model Views,” International Journal of Computer Vision, vol. 67, no. 2, pp. 159-188, 2006.

[37] N. Abbas, “Graph Clustering: Complexity, Sequential and Parallel Algorithms”, Department of Computing Science, 1995.

[38] T. Hastie, R. Tibshirani and J. Friedman, “The Elements of Statistical Learning: Data Mining, Inference and Prediction,” Springer on Mathematical Intelligencer, vol. 27, no. 2, pp. 83-85, Jun. 2005.

[39] L. Vincent and P. Soille, “Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations,” IEEE Trans. on Pattern Analysis and Machine Intelligence, vol.13, no.6, pp. 583-598, Jun. 1991.

[40] J. Roerdink and A. Meijster, “The Watershed Transform: Definitions, Algorithms and Parallelization Strategies,” Fundamenta Informaticae, vol. 41, no. 1-2, pp. 187-228, 2000.

[41] J.W. Jang, H. Kim and H.J. Lee, "Block-based Marker Controlled Watershed Transform Using Motion Information", ITC-CSCC 2012, Jul. 2012.

[42] A.N. Moga and M. Gabbouj, "Parallel Image Component Labelling with Watershed Transformation," IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 19, no. 5, pp. 441-450, May 1997.

[43] S.Y. Chien, Y.W. Huang, and L.G. Chen, “Predictive Watershed: A Fast Watershed Algorithm for Video Segmentation,” IEEE Trans. on Circuits and Systems for Video Technology, vol.13, no.5, pp. 453- 461, May 2003.

[44] J. Kim, H.J. Lee, T.H. Lee, M. Cho, and J.B. Lee, "Hardware/Software Partitioned Implementation of Real-time Object-oriented Camera for Arbitrary-shaped MPEG-4 Contents," in Proc. of IEEE/ACM/IFIP Workshop on Embedded Systems for Real Time Multimedia 2006, pp. 7-12, Oct. 2006.

[45] B.J. Mealy, “Scanning Order Dependencies in Watershed Transform”, Technical Report UCSC-CRL-02-37, Dec. 2002.

[46] D. Wang and C. Labit, “Morphological Spatio-Temporal Simplification for Video Image Segmentation,” Signal Processing: Image Communication, vol. 11, no. 2, pp. 161-170, Dec. 1997.

[47] T.H. Kim, K.M. Lee and S.U. Lee, “A Unified Probabilistic Approach to Feature Matching and Object Segmentation,” 20th International Conference on ICPR 2010, pp. 464-467, Aug. 2010.

Abstract

Due to a popular use of personal storage devices and surveillance systems, the amount of stored image data is rapidly increasing. Therefore, a demand for an image recognition system using stored image data has been increased in a variety of applications such as object classification, object recognition, object tracking and so on. However, its excessive computation makes it difficult to be processed in real time by a device with limited resource such as a mobile phone. In this paper, several methods are proposed to perform object recognition at high speed.

First, this paper presents an adaptive keypoint generation for SIFT hardware implementation. In the image recognition system using local feature, SIFT (Scale-Invariant Feature Transform) has been proven to be one of the most robust local feature descriptors for various image transformations. To speed up SIFT generation, it is important to reduce the number of keypoints. The previous method apply the same threshold value regardless of the image characteristics. This paper addresses this problem and proposes a block-based keypoint adjustment controlling the keypoint generation using the characteristics of the block. The proposed method is capable of maintaining the pipeline structure of SIFT hardware and can generate a meaningful keypoint distribution. Furthermore, it is possible to reduce the execution time.

Second, feature matching method based clustering requires a large amount of computation because it needs to consider all correspondences of the image. This paper proposes a region-constrained feature matching in which an image is segmented into small regions and feature correspondences are clustered inside each region. The proposed region-constrained clustering dramatically reduces the execution time while it achieves a similar matching accuracy.

Third, watershed transform is difficult to process in parallel because it sequentially uses the values in the image. The proposed algorithm is processed in a block-based manner such that an image is decomposed into blocks and each block is processed independently of the other blocks. The block-based watershed transform can further reduce the computation by examining the necessity of flooding operations for each block and avoiding them when they are not necessary. The proposed method provides the same segmentation accuracy.

Keywords: SIFT, FAST, Hardware, SoC, Feature matching, Segmentation

Student number: 2008-30240

공학박사 학위 논문

영역 분할을 활용한 Hardware 기반 고속 Local Feature Matching

Hardware-based Fast Local Feature Matching
Using Region Segmentation

2014년 2월

서울대학교 대학원
전기·정보공학부
장 정 환

영역 분할을 활용한 Hardware 기반 고속 Local Feature Matching

Hardware-based Fast Local Feature Matching Using Region Segmentation

지도교수 김 수 환

이 논문을 공학박사 학위논문으로 제출함

2014년 2월

서울대학교 대학원

전기·정보공학부

장 정 환

장정환의 공학박사 학위 논문을 인준함

2014년 2월

위 원 장: 채 수 익 (인)

부위원장: 김 수 환 (인)

위 원: 최 진 영 (인)

위 원: 이 혁 재 (인)

위 원: 김 진 성 (인)

초 록

최근 디지털 저장 장치와 개인 영역에서도 surveillance system 이 빠르게 늘어남에 따라서, 저장된 이미지 데이터의 양이 급격하게 늘어나고 있으며, 저장된 데이터 내에서 유사 이미지 검색, 분류, 인식 등의 응용이 점차 증가하고 있다. 이와 같은 영상 인식 시스템에서 SIFT local feature 를 이용한 영상 인식 방법은 다양한 영상의 변화에 강인한 특성을 가지기 때문에, 최근 들어 모바일, robot navigation, object recognition 등 여러 분야에서 활용도가 점차 높아지고 있다. 영상 인식 시스템은 많은 양의 연산을 필요로 하는 단계들을 거쳐 수행되기 때문에, mobile 과 같은 제안된 resource 를 가지는 환경에서 고속으로 처리하는데 어려움이 있다. 본 연구에서는 고속으로 영상인식을 수행하기 위한 방법들을 제안한다.

SIFT 하드웨어 에서의 속도 향상 방법으로 adaptive SIFT keypoint 생성방법을 제안하였다. Keypoint 의 수는 전체 하드웨어의 수행시간에 영향을 미치는 요소이다. 이를 조절하기 위해 본 연구에서는 블록단위로 영상의 복잡도를 예측하고, 블록의 contrast threshold 값을 달리 적용하여, keypoint 의 수를 조절하는 방법을 사용하였다. 제안한 방법은 SIFT 하드웨어의 pipeline 구조 유지가 가능하고, keypoint 발생 분포면에서 유리한 성능을 보인다. 또한 이를 통해 전체 수행시간을 단축할수 있음을 확인하였다.

SIFT 에서 생성된 descriptor 사이에 Euclidean distance 의 유사성을 사용한 matching 방법은 많은 수의 잘못된 correspondence 를 포함하게 된다.

초기의 correspondence 로부터 inlier 와 outlier 를 구분하기 위한 clustering 기반의 feature matching 방법은 이미지내에 모든 correspondence 를 대상으로 clustering 을 수행하므로, 많은 수의 연산이 필요하다. 본 연구에서는 영역 단위로 clustering 을 수행하는 방법을 제안하고, 영역내의 clustering 결과의 집합으로 전체의 결과를 표시하였다. Clustering 을 수행하는 영역은 이미지의 segmentation 정보를 사용하여, 유사한 특성의 correspondence 들을 포함하는 인접 segment 들을 합하여 구성하였다. 제안한 방법은 동등 수준의 성능에서도 수행시간을 크게 단축할수 있음을 확인하였다.

Watershed segmentation 방법은 이미지내의 값들을 순차적으로 사용하기 때문에 병렬처리가 어렵다. 본 연구에서는 이미지를 블록 단위로 나누고, 블록간 독립적인 연산을 가능하게 하는 방법을 제안하였다. 블록 단위 연산에서 상호의존적인 부분은 블록 경계면에서의 gradient 값과 인접해 있는 segment 들간의 root 값을 통해 처리하였다. 또한, 하나의 segment 로 예측되는 블록들에 대해선 watershed 의 일부 연산을 수행하지 않도록 하였다. 제안된 방법은 기존 watershed 의 결과와 동일한 수의 segment 를 가지는 것을 확인하였고, 블록별 병렬처리를 통해 수행시간이 줄어드는 것을 확인하였다.

주요어: SIFT, FAST, 하드웨어, SoC, Feature matching, Segmentation
학번: 2008-30240

차 례

그림 목차

표 목차

제1장 서론

1.1 연구 배경

최근 디지털 저장 장치와 CCTV 의 증가로 인하여 저장된 이미지 데이터의 양이 급격하게 증가하고 있으며, 저장된 데이터 내에서 유사 이미지 검색, 분류, 인식 등의 응용이 점차 증가하고 있다.

여러 영상 인식 시스템 중 local feature 를 이용한 영상 인식 방법들은 noise 나 light variation, viewpoint 변화 등과 같은 영상 변화에 강인한 특성을 가지고 있기 때문에, 최근 들어 컴퓨터 비젼 분야에서 많이 연구되고 있다[3][7].

Local feature 기반의 영상 인식 시스템은 그림 1.1 과 같이 구성할 수 있다. Local feature 를 이용한 영상 인식 과정은 feature extraction 과정과 feature matching 과정으로 구성되어 있다.

Feature extraction 단계에서는 영상에서 주목할 만한 feature point 들을 추출하여 feature 주변의 영상을 기술하는 feature descriptor 로 만드는 과정까지를 포함한다. 이 단계 이후의 연산은 입력 이미지가 아닌, feature descriptor 를 통해 이루어진다. 이 단계에서는, 필요한 정보만을 추출하기 위해 수많은 입력 픽셀에 대한 연산을 수행해야 하기 때문에 매우 많은 연산량이 필요하다. 또한, 이 단계는, 필요한 정보만을 남기고 나머지는 버리기 때문에 그 결과에 따라 물체 인식 결과가 크게 바뀔 수 있다.

Feature matching 단계에서는 feature point 들간에 상관성을 조사하여 feature 들간에 대응 관계를 찾는 과정을 수행한다[4]. 대응 관계를 찾는 과정은 feature extraction 통해서 계산된 feature descriptor 간에 유사성을 통해 판단한다.

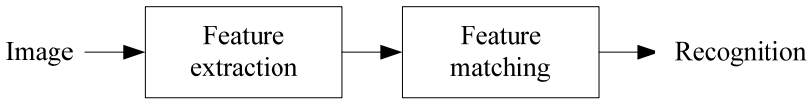


그림 1.1 General recognition system

특히 최근 들어 많은 양의 연산을 필요로 하는 영상 처리 방법들을 모바일에서 수행하기 시작하면서, 위와 같은 인식 시스템에 대한 알고리즘을 효과적으로 구현하여 속도를 높이는 것이 중요해 지고 있다. 모바일 환경에서 주로 이루어지는 영상 처리 방법들로는 영상 검색, 얼굴 인식, 파노라마 이미지 생성 등이 있다. 또한 최근 검색 엔진 등에서도 영상 검색 기술이 적용되고 있다[1][2][5][6][8]. 따라서, 이러한 local feature 기반의 recognition system 을 실시간으로 처리하기 위한 방법들에 대한 연구가 많이 진행되고 있다.

본 연구에서는 local feature 기반의 recognition system 을 수행하기 위해서 단계별로 각각의 특성을 분석하고, 효과적으로 연산하기 위한 방법들을 제안한다.

1.2 연구 내용

본 연구에서는 효율적으로 물체를 인식하기 위한 방법들을 연구하였다. 이 중 SIFT 는 여러 가지 방법 중, 가장 뛰어난 성능을 보이는 feature extraction 방법으로 알려져 있다[12][13]. 하지만 SIFT 는 많은 연산량을 필요로 하기 때문에 실시간 처리를 위한 방법으로는 많은 어려움이 있다. 이러한 문제점을 개선하고자, 본 연구에서는 하드웨어로 구현된 keypoint (feature point) 와 descriptor generation 과정을 이용하였고, 많은 수의 keypoint 로 인해서 길어진 수행 시간을 줄이기 위해서, 하드웨어에 적합한 블록 기반의 keypoint 수 조절 방법을 제안하였다. 이와 같은 방법들을 이용하여, 본 연구에서는 전체 하드웨어 속도를 향상하는 방법을 제안하였다.

Descriptor 를 사용해서, local patch 의 유사성만으로 상관성을 추출하면, 많은 수의 잘못된 correspondence 가 발생하게 된다. 따라서, correspondence 가운데, correct correspondence(inlier) 와 wrong correspondence(outlier) 를 구분할 필요가 있다. 본 연구에서는 다양한 영상 변형에서도 효과적으로 inlier 와 outlier 를 구분하는 clustering 기반의 matching 방법을 사용하였다. 그러나 clustering 기반의 알고리즘은 correspondence 의 수에 따라서 급격하게 처리할 연산이 늘어나는 문제점을 가지고 있다. 따라서 본 연구에서는 clustering 이 될 가능성이 높은 correspondence 들을 region 단위로 나누어, region 내에서 clustering 을 수행하는 방법을 제안하였다.

또한 region 단위로 clustering 을 수행하기 위해서 필요한 segmentation 을 효과적으로 수행하기 위해서, 블록 기반의 watershed 방법을 제안하였다.

제안된 방법의 검증은 그림 1.2 와 같이 하드웨어상에서 keypoint 수 조절 방법을 적용하여 이에 대한 성능 향상을 확인하였고, 소프트웨어 구현을 통해 region 기반의 clustering 알고리즘과 segmentation 알고리즘의 성능 검증을 수행하였다.

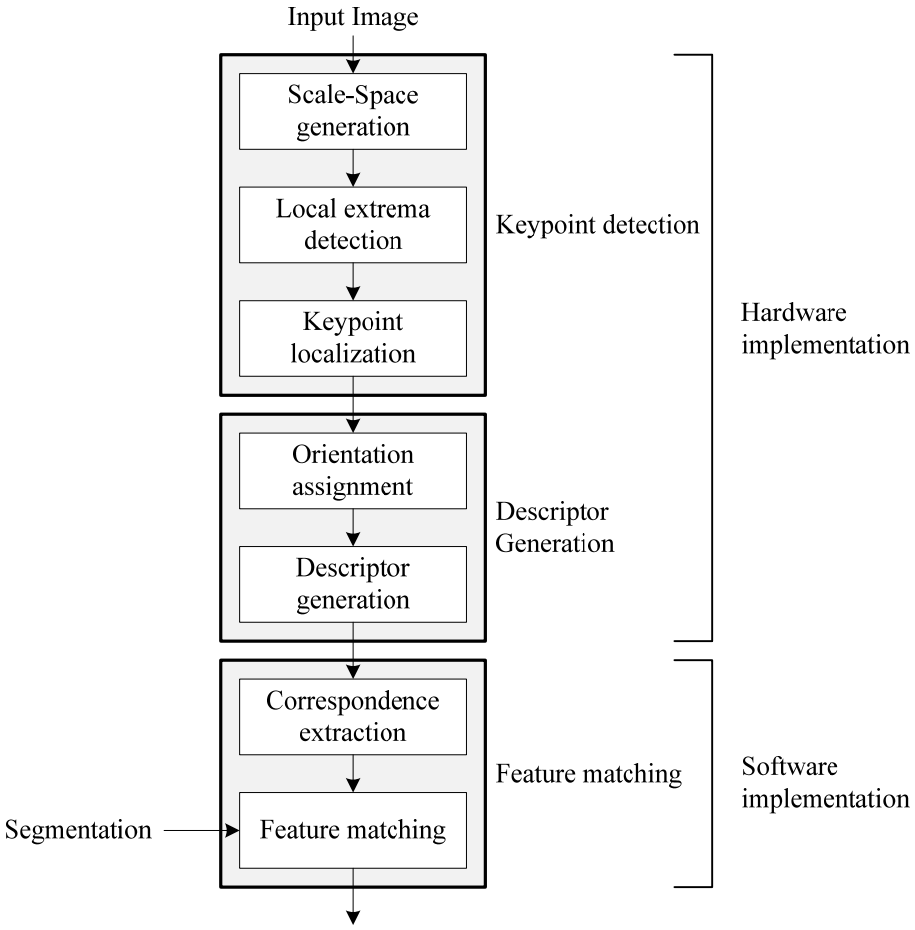


그림 1.2 시스템 구성 및 검증 flow

1.3 논문 구성

본 연구의 나머지 구성은 다음과 같다. 2장에서는 본 연구의 기본이 되는 이론과 기존 연구를 소개 한다. 3장에서 adaptive keypoint generation 을 위한 하드웨어 구조를 제안하며, 4장에서는 생성된 keypoint 로부터 inlier 와 outlier 를 구분하는 방법으로 사용하는 clustering 을 효과적으로 구현하는 region constrained feature matching 방법을 제안한다. 5장에서는 4장에서의 region 을 구성하는 단위로서 사용되는 watershed segmentation 을 병렬 처리가 가능한 블록 단위로 수행하는 방법을 제안한다. 마지막으로 6장에서 결론을 맺는다.

제2장 관련 연구

이 장에서는 본 연구의 배경 이론인, SIFT 의 keypoint 와 descriptor 생성 알고리즘에 대해 설명하고, 이에 대한 문제점을 설명한다. 또한 feature matching 을 위한 기존 연구와 그에 대한 문제점을 설명한다.

2.1 SIFT (Scale-Invariant Feature Transform)

Feature detector 는 이미지에서 corner, blob, edge 와 같은 이미지 내의 특징된 부분을 추출하는 방법을 의미한다. Feature detector 들에서는 이러한 특징된 부분들을 표현하기 위해서 특징의 위치를 표현하는 keypoint 와 그 위치에서의 주변 영상의 특징을 표현하는 descriptor 을 사용한다.

이중 SIFT 는 Lowe[9] 가 제안한 local feature 의 한 종류로, 여러 local feature 중 가장 성능이 우수한 방법으로 알려져 있다. SIFT 는 크게 keypoint detector 와 feature 주변의 영상을 기술하는 descriptor 를 생성하는 부분으로 구성되어 있다. 또한 [9] 논문에서는 SIFT descriptor 간 유사성을 비교하여 correspondence 를 생성하는 과정까지를 포함하고 있다. 그림 1.2 와 같이 keypoint detection 단계에서는 scale-space generation, local extrema detection, keypoint localization 의 세부적인 연산으로 구성되어 있으며, descriptor 생성 단계에서는 orientation assignment, descriptor generation 의 세부적인 연산으로 구성되어 있다. 이후 절에서는 각각의 세

부 부분에 대해서 설명한다.

2.1.1 Scale-space generation

SIFT 에서는 scale 변화에 따른 keypoint 를 추출하고, 이를 통해, scale 이 변화된 영상에서도 이미지간 matching 이 가능하도록 하고 있다. 이를 위해 이미지의 scale-space 를 생성하고, 여러 scale 에서 keypoint 를 생성하게 된다[10].

Scale-space generation 단계에서는 입력 이미지의 Gaussian filtering 과정을 통해서, Gaussian filtering 된 이미지를 생성하고, 이를 통해, DoG(Difference Of Gaussian) 이미지를 생성한다.

먼저 입력 이미지 $I(x,y)$ 로부터 Gaussian filtering 된 이미지, $L_i(x,y)$ 는 다음과 같은 식을 통해 계산된다.

$$L_i(x,y) = G(x,y,\sigma_i) * I(x,y) \quad for\ i = 0,1,\dots,S+2 \qquad (2.1)$$

$$G(x,y,\sigma_i) = \frac{1}{2\pi\sigma_i^2} e^{-\frac{x^2+y^2}{2\sigma_i^2}} \quad where\ \sigma_i = \sigma_0 \cdot 2^{\frac{i}{S}} \qquad (2.2)$$

입력 이미지에 대한 Gaussian filtering 된 이미지는 식 (2.1)와 같이 입력 이미지와 Gaussian filter 의 convolution 과정을 통해서 계산된다. 이때, 사용되는 Gaussian kernel 은 식 (2.2)와 같이 나타난다.

Gaussian filtering 된 이미지는 Gaussian kernel 에 따라서 다르게 생성 되는데, Gaussian kernel 의 scale σ_i 로 생성된 filtering 된 이미지를 $L_i(x,y)$ 로 표시한다. Octave 는 scale-space 를 구성하기 위해서 down-sampling

하고 나서, 다음 down-sampling 까지의 Gaussian filtering 된 이미지들의 집합으로 구성된다. 식 (2.2) 에서 S 는 octave 당 scale 의 개수이다. 본 연구에서는 S 값으로 3 을 사용하였다. Octave 당 3개의 scale 에 대해서, 각각 feature 를 생성하기 위해서는 S+2 의 DoG 이미지가 필요하게 된다. 또한 DoG 이미지는 S+3 개의 Gaussian filtering 된 이미지로부터 만들어진다. 따라서, S 가 3 일 경우 6 개의 Gaussian filtering 된 이미지를 만들어야 한다.

예를 들어 그림 2.1 과 같이 3x3 크기의 Gaussian filter 를 사용하여, 1 픽셀의 결과를 생성하기 위해서, 3x3 크기의 입력 영상과 3x3 크기의 Gaussian filter 의 convolution 연산을 수행하게 된다[21].

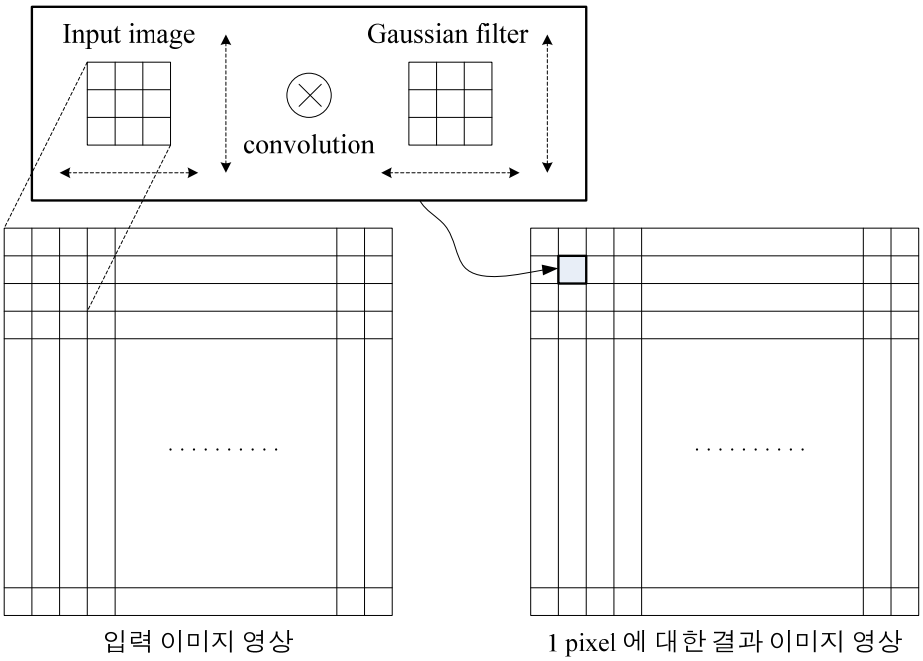


그림 2.1 Gaussian filtering 된 이미지 생성을 위한 연산 과정

DoG 이미지는 두 개의 Gaussian filtering 된 이미지의 차 영상을 의미한다. DoG 이미지는 식 (2.3) 을 통해 계산이 가능하다.

$$D_i(x,y) = \big(G(x,y,\sigma_{i+1}) - G(x,y,\sigma_i)\big) * I(x,y) \\ = L_{i+1}(x,y) - L_i(x,y) \tag{2.3}$$

표 2.1 는 S 가 3 일 경우 Octave 0 에 대해서 Gaussian filter 의 scale 에 따른 Gaussian filtering 된 이미지와 DoG 이미지간의 관계를 보여준다. S 가 3 일 경우 5개의 DoG 이미지가 생성됨을 볼 수 있다.

표 2.1 Scale 에 따른 filtered image 와 DoG 영상 생성

Octave 0						
Filter scale	σ_0	$\sigma_0 \cdot 2^{1/3}$	$\sigma_0 \cdot 2^{2/3}$	$\sigma_0 \cdot 2$	$\sigma_0 \cdot 2^{4/3}$	$\sigma_0 \cdot 2^{5/3}$
Filtering 된 이미지	L_0	L_1	L_2	L_3	L_4	L_5
DoG image		D_0	D_1	D_2	D_3	D_4

그림 2.2 는 Oxford dataset 의 Graffiti 이미지를 사용하였을 때, Octave 0,1,2 에서 생성된 Gaussian pyramid 이미지를 보여준다. Octave 1 은 원본 영상을 $2\sigma_0$ 의 Gaussian filter scale 로 filtering 한 L_3 의 영상을 가로 세로 방향으로 각각 1/2 씩 down-sampling 하여, 두 번째 octave 의 L_0 영상을 만든다.

이때 적용되는 Gaussian filter 는 그림 2.2 의 상단 이미지와 같이 octave 내에서 filter scale 에 따라 다른 크기의 Gaussian filter 가 사용된다. 그림

2.3 은 Octave 0 에서 Gaussian filtering 된 이미지로부터 생성된 DoG 를 보여준다. 그림에서와 같이 DoG 이미지는 서로 다른 Gaussian filter 의 scale 로 생성된 이미지의 차이기 때문에, edge 나 blob 이 존재하는 부분에서 local extrema, 즉 local minimum 이나 local maximum 이 발생한다. 그림에서의 DoG 이미지는 시각적으로 강조해서 보여주기 위해 histogram equalization 과정을 수행하여, 이미지의 contrast 를 증가시킨 이미지이다.

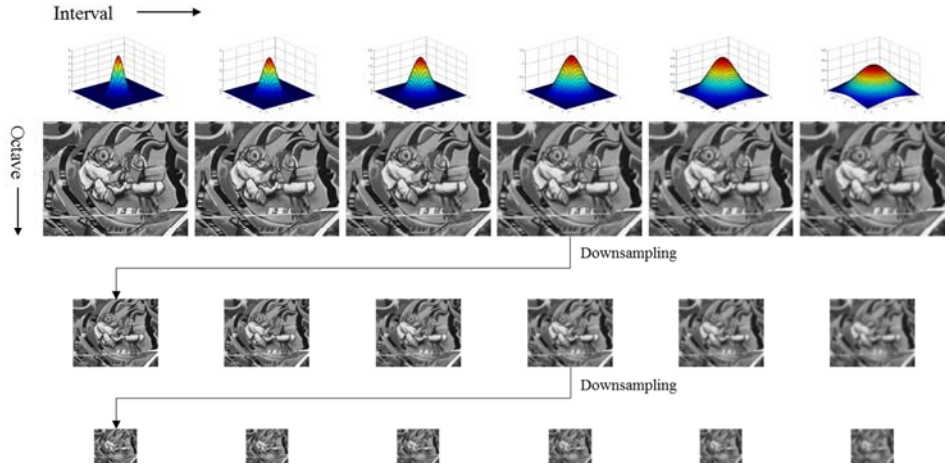


그림 2.2 Gaussian pyramid images

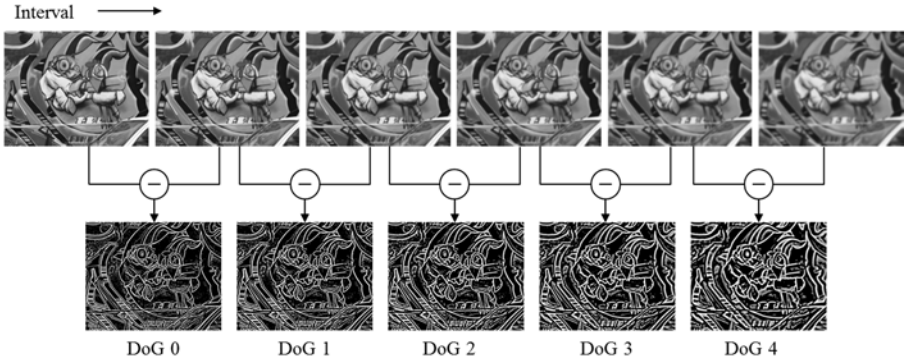


그림 2.3 Octave 0 DoG images

2.1.2 Local extrema detection

이미지의 변화에 대해 invariant 한 keypoint를 추출하기 위해서, local extrema detector 단계에서는 그림 2.4 과 같이 인접한 3개의 DoG 이미지를 사용한다. 현재 픽셀이 속한 DoG 이미지와 인접한 2 개의 DoG 이미지를 사용하여 구성한 3x3x3 형태의 픽셀 집합들에서 상하 좌우로 인접한 총 26개의 픽셀을 비교한다. 따라서, 5개의 DoG 이미지로부터 3 개의 keypoint detection 를 위한 pyramid 가 구성가능하다. 이때 현재 픽셀이 local minimum 이나 local maximum 인 경우 특징점을 위한 후보로 추출된다.

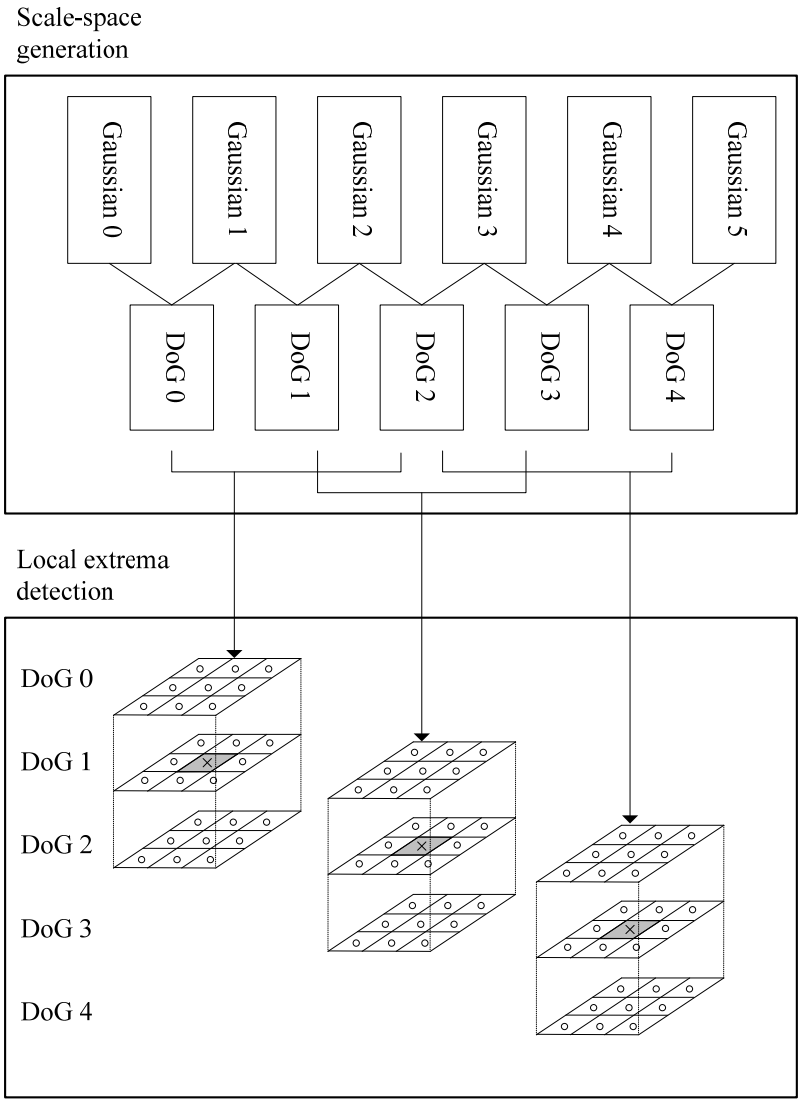


그림 2.4 Local extrema detection

2.1.3 Keypoint Localization

Keypoint Localization 단계에서는 keypoint 의 위치를 세부적으로 조정하거나, low contrast 값을 가지는 경우 또는 edge 주변에 위치한 keypoint 를 제거하는 과정을 수행한다. 이전 단계에서의 local extrema 을 sub-pixel, sub-scale 단위까지 세부적으로 찾기 위해서, 주변 DoG 값을 이용한 interpolation 을 수행한다. 이것은 식 (2.4)와 같이 DoG 함수의 Taylor 시리즈를 이차항까지 전개한 수식을 사용한다. 여기서 $\mathrm{D(x)}$ 는 해당 keypoint 의 후보점을 원점으로 하여 계산한다. 또한, 벡터 $\mathrm{x} = [x, y, \sigma]^T$ 는 후보점에서의 sub-pixel 단위의 offset 을 의미한다.

$$D(\mathrm{x}) = D + \frac{\partial D^T}{\partial \mathrm{x}}\mathrm{x} + \frac{1}{2}\mathrm{x}^T\frac{\partial^2 D}{\partial \mathrm{x}^2}\mathrm{x} \tag{2.4}$$

Extrema 의 위치 $\hat{\mathrm{x}}$ 은 식을 x 에 대해서 미분한 후 0 으로 두면, 식 (2.5) 와 같은 결과를 생성하는데, 이때의 x 를 $\hat{\mathrm{x}}$ 로 한다.

$$\hat{\mathrm{x}} = -\frac{\partial^2 D}{\partial \mathrm{x}^2}^{-1}\frac{\partial D}{\partial \mathrm{x}} \tag{2.5}$$

이 값이 식 (2.4) 로 표현된 interpolation 식이 극값이 되는 sub-pixel offset 이다. 만약 식 (2.5) 의 값이 벡터 x 의 어떤 방향에서라도 크기가 0.5 을 넘는 것이 있으면, keypoint 후보점의 위치를 조정하게 된다.

또한 식 (2.6) 으로 표현된 extrema 위치에서 함수값이 low contrast 를 가지는 불안정한 위치의 extrema 값은 제거하게 된다. 논문[9] 에서는

$|D(\hat{x})| < 0.03$ 의 값을 가지는 keypoint 의 위치를 keypoint 후보에서 제외한다.

$$D(\hat{x}) = D + \frac{1}{2}\frac{\partial D^T}{\partial x}\hat{x} \tag{2.6}$$

여기서 사용되는 contrast threshold 값은 최종적으로 선택되는 keypoint 의 수에 관련이 있는 변수이다. 또한 이와 같은 threshold 값은 하나의 이미지에 대해서, 동일하게 적용되는 값이다. 따라서, contrast 가 강한 영역 주변의 keypoint 들은 많이 유지되고, contrast 가 약한 영역 주변의 keypoint 는 제거된다.

2.1.4 Orientation assignment

Orientation 의 변화에 대해 영향을 받지 않는 descriptor 를 생성하기 위해서, 해당 keypoint 에서의 dominant 한 orientation 을 계산하게 된다. 각 keypoint 에 해당하는 scale 의 Gaussian filtering 된 이미지 $L(x, y, \sigma)$ 을 사용하여, keypoint 주변의 gradient magnitude $m(x,y)$와 orientation $\theta(x,y)$ 을 계산한다. Keypoint 위치에서의 gradient 는 식 (2.7) 식 (2.8)과 같이 주변 픽셀 값의 뺄셈으로 계산된다.

$$\Delta_x = \frac{1}{2}\big(L_i(x+1,y) - L_i(x-1,y)\big) \tag{2.7}$$

$$\Delta_y = \frac{1}{2}\big(L_i(x,y+1) - L_i(x,y-1)\big) \tag{2.8}$$

$$m(x,y) = \sqrt{\Delta_x^2 + \Delta_y^2} \qquad \theta(x,y) = tan^{-1}\left(\frac{\Delta_y}{\Delta_x}\right) \tag{2.9}$$

2.1.5 Descriptor generation

그림 2.5 는 keypoint descriptor 를 계산하는 과정을 보여 준다. 이전 절에서 계산한 orientation 을 통해 local patch 의 dominant orientation 을 계산한다. 이를 위해 keypoint 주변 local patch 에서 orientation histogram 을 사용하는데, orientation histogram 은 local patch 내의 Gaussian filtering 된 이미지 $L(x, y, \sigma)$ 의 gradient orientation 으로 생성된다. Orientation histogram 은 360 도를 표현하기 위해서 36 개의 bin 으로 나뉘어 있다. Histogram 을 생성하기 위해서 keypoint 를 중심으로 keypoint 의 scale 의 1.5σ 에 해당하는 영역 내의 값을 사용하게 된다. 영역 내의 gradient 값은 해당 위치에서의 scale 의 1.5σ 에 해당하는 Gaussian-weighted circular window 에 의해 가중치가 가해진 후 그 값이 orientation 이 속하는 bin 에 축적된다.

Orientation histogram 에서의 peak 값은 local gradient 의 dominant direction 을 나타낸다. 선택적으로 local peak 값의 80% 이상인 direction 에 대해서, 추가적인 dominant orientation 으로 설정하기도 한다. 따라서, 이러한 경우엔 동일한 위치, scale 을 가지고, 다른 orientation 을 가지는 복수 개의 keypoint 가 가능하게 된다.

이후, dominant orientation 으로 선택된 가장 큰 각도를 기준으로 정규화하여, 회전에 무관한 orientation 으로 변환된다. 그것을 다시 8 개의 각도로 통합하고, 그것을 히스토그램으로 만들어 한 영역의 descriptor 로 사용한다

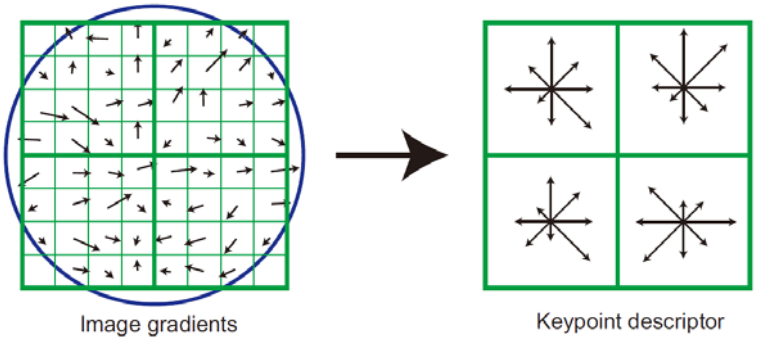


그림 2.5 Keypoint descriptor 생성 방법

그림 2.5 는 keypoint 주변의 descriptor 를 생성하는 예를 보여준다[9]. Gradient 이미지는 8x8 sample 들로 구성되어 있고, 이를 통해 2x2 descriptor array 를 생성하는 과정을 보여준다. 본 연구에서는 16x16 array 를 사용해서, 4x4 descriptor 를 생성하였다. 각 descriptor array 는 8 개의 bin 을 가진다. 따라서 실험에서는 4x4x8 = 128 개의 feature vector 를 가지는 descriptor 를 생성하게 된다.

2.1.6 Descriptor correspondence

본 단계에서는 이전 절에서 생성된 descriptor 들을 통해 이미지간 matching 을 수행한다. 본 연구에서는 두 개의 이미지에서 descriptor 간의 유사성을 통해서, 가장 유사한 descriptor 로 찾은 descriptor 쌍을 correspondence 라고 하였다. 또한, correspondence 내에는 많은 수의 incorrect correspondence 를 포함하기 때문에, correspondence 로부터 correct correspondence 를 찾는 과정을 matching 으로 사용하였다.

각 keypoint 의 descriptor 에 대해 correspondence 를 찾는것은 다른 이

미지의 keypoint database 중 가장 유사한 descriptor 를 찾는 것이다. 두 개의 descriptor vector $\mathbf{u}, \mathbf{v}$ 간에 유사성은 Euclidean distance 로 표현하고, 식 (2.10)과 같이 정의된다.

$$d(\boldsymbol{u}, \boldsymbol{v}) = \left(\sum_i (u_i - v_i)^2 \right)^{1/2} \tag{2.10}$$

SIFT 의 descriptor 는 128 descriptor vector 를 갖기 때문에, 128 차원의 Euclidean distance 로 계산된다[9]. 이와 같이 계산된 Euclidean distance 를 사용하여 correspondence 를 구성하는데는 몇 가지 방법들이 있다. 먼저, NN(Nearest Neighbor) 방법은 최소의 Euclidean distance 를 갖는 descriptor 를 선택하는 방법이다. 또한 NNDR(Nearest Neighbor Distance Ratio)을 사용하는 방법은 식 (2.11) 과 같이 nearest neighbor 를 가지는 descriptor 의 Euclidean distance (d_1) 과 second nearest neighbor 를 가지는 descriptor 의 Euclidean distance(d_2) 간의 distance ratio 를 사용하는 방법이다.

$$NNDR = \frac{d_1}{d_2} \tag{2.11}$$

따라서 NNDR 값은 1 보다 작은 값을 가지게 되고, NNDR 의 값이 작을수록, 즉 두 descriptor 간의 차이가 분명히 발생하는 경우에 좋은 correspondence 로 판단한다.

2.2 FAST (Features from Accelerated Segment Test)

FAST (Features from Accelerated Segment Test)[22]은 이미지내에 존재하는 corner 를 keypoint 로 찾는 detector 방법이다. 특히 FAST 는 빠른 속도로 코너 검출이 가능하다. FAST 에 의한 코너 검출 방법은 다음과 같은 단계를 거쳐서 생성된다.

먼저 keypoint 를 찾기 위해 전체 이미지에서 corner 로 판단할 후보 위치 p 의 주변을 원 형태로 16개 픽셀을 조사한다. Corner 후보 p 에 대해서 주변 조사를 위한 픽셀 위치는 그림 2.6 과 같이 구성된다[22]. 16개의 픽셀을 각각 후보 p의 밝기 정도 I_p와 미리 설정한 threshold 값 t를 더한 것 보다 밝으면 그 픽셀은 brighter로 두고, 값 t를 뺀 것 보다 어두우면 그 픽셀은 darker로 둔다. 그 외의 픽셀은 similar로 둔다. 이를 수식으로 나타내면 식 (2.12)과 같다. 원에 해당하는 픽셀의 위치에 따라 값 x를 가지고 밝기는 $I_{p\to x}$ 로 둔다.

$$S_{p\to x} = \begin{cases} d, & I_{p\to x} \le I_p - t \\ s, & I_p - t \le I_{p\to x} \le I_p + t \\ b, & I_p + t \le I_{p\to x} \end{cases} \tag{2.12}$$

후보 p의 주변 16개의 픽셀중 연속된 n개 이상의 픽셀이 darker이거나 brighter이면 후보 p를 corner로 설정한다. Corner가 되기 위한 최소 n의 값은 9이고, 충분한 corner를 검출하기 위한 최대 n의 값은 12이다. 이를 반복하여 전체 이미지에서 corner를 검출해 낸다.

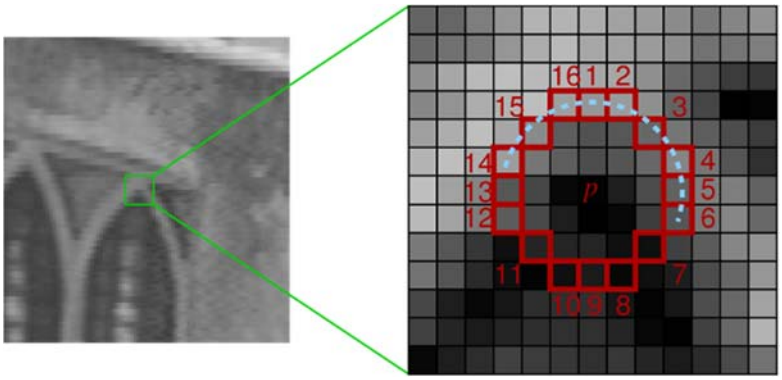


그림 2.6 FAST corner detector

검출해 낸 corner 들을 최종 keypoint 로 선정하기 전에, 위와 같은 조건을 만족하는 경우의 corner 들을 추출하면 한 개의 corner에 keypoint 가 여러 개 검출된다. 따라서 비슷한 영역의 존재하는 keypoint 의 개수를 줄이는 알고리즘이 필요한데, 이 알고리즘을 non-maximal suppression 이라고 한다. Non-maximal suppression 알고리즘은 후보 p마다 score를 매긴 후 1 픽셀위치 만큼 차이가 나는 두 후보 p의 score를 비교하여 score가 작은 후보 p는 corner에서 제외하는 방법을 사용한다. Score 는 식 (2.13) 와 같이 계산된다.

$$score = max \begin{cases} \sum \left(p \text{ 의주변값} - p\right) & if \left(p \text{ 의주변값} - p\right) > t \\ \sum \left(p - p \text{ 의주변값}\right) & if \left(p - p \text{ 의주변값}\right) > t \end{cases} \quad (2.13)$$

위와 같이 FAST detector 를 통해서, 추출한 keypoint 들은 영역 내에 발생한 keypoint 의 수를 합해서, 영역 내에 이미지의 특성을 예측하기 위한 방법으로 사용된다.

2.3 이전 연구

일반적인 recognition 시스템은 1 장에서 설명한 바와 같이 크게 keypoint 를 찾고, 이를 기술하는 descriptor 를 만드는 과정(feature extraction)과 이를 통해 matching(feature matching) 하는 단계로 구성되어 있다. 이 절에서는 각각의 단계에서 효과적인 연산을 수행하기 위해서, 기존에 연구되었던 내용들을 소개하고, 기존 방법의 문제점을 살펴본다.

2.3.1 Keypoint selection and spatial distribution

Keypoint detection 단계에서 생성된 많은 수의 keypoint 는 이후 단계의 연산 과정인 descriptor 생성 단계와 matching 단계에서 keypoint 수에 비례한 연산과정을 필요로 한다. 이미지에서 생성되는 많은 수의 keypoint 는 이미지의 특징을 많이 추출한다는 측면에서는 좋지만, 이후 단계의 연산에 부담이 될 뿐만 아니라, 동시에 정확한 위치를 찾아야 하는 응용에서는 비슷한 위치에 발생하는 여러 개의 keypoint 로 인해 잘못 correspondence 가 생성될 가능성이 높아진다. 또한, 영역을 찾는 응용의 경우에는 과도하게 많은 수의 keypoint 가 특정 영역에 존재하기 보다는 keypoint 의 분포가 고르게 되어 있는 경우가 더 유리하다. 이와 같은 필요성에 의해서, keypoint 수를 조절하기 위한 연구들이 진행되어 왔다[27].

먼저 단순히 발생된 keypoint 중 N 개의 가장 강한 특성을 가지는 feature point 를 찾는 방법이 있다[23]. 이와 같은 방법은 가장 특성이 강한 영역 주변의 keypoint 만 선택되게 된다. 따라서, 특정 영역에만 이미지를 표현하는 keypoint 가 남게 되기 때문에, 상대적으로 특성은 낮지만, 필요한

위치를 기술할 keypoint 가 없어지는 문제가 발생한다. 따라서 특히 image mosaicking, robot navigation, scene recognition 응용에서와 같이 keypoint 간 상대적 위치를 고려하는 응용에서는 특정영역에만 keypoint 가 분포되어 있는 경우엔 불리하게 된다[23][24]. 이와 같은 경우에서도 볼 수 있듯이 단순히 keypoint 의 수뿐만 아니라, keypoint 의 분포 또한 고려가 되어야 한다. 이를 개선하기 위해서, 전체 이미지에 일정하게 분포된 keypoint 를 선택하기 위해서, ANMS(Adaptive Non-Maximal Suppression) 방법을[25] 사용하여, 발생된 keypoint 에 대해서, 일정 범위 내에서 가장 강한 특성을 가지는 keypoint 를 선택하는 방법이 있다. 또, 발생된 keypoints 중, 사전에 계산된 영역에서 하나씩만 선택하는 방법[26][29], 발생된 keypoint 를 트리 형태의 구조를 이용해서 선택하는 방법[28]등을 사용해서 전체 이미지 내에서 발생하는 keypoint 를 줄이는 방법에 대한 연구들이 진행되었다. 이와 같은 방법들은 keypoint 의 수를 조절하여, 이후 단계에 수행되는 descriptor 생성 단계나 matching 단계의 수행 시간을 줄이는데 효과적으로 사용 가능할 뿐만 아니라, keypoint 들이 전체적으로 고르게 분포되어서 전체 이미지를 효과적으로 표현할 수 있다.

그러나 이와 같은 방법은 발생된 keypoint 결과에 의존성을 가지고 있다. 즉, keypoint 가 생성된 이후 발생된 keypoint 의 분포에 대해서 적용이 가능한 방법들이다. 따라서, 하드웨어로 구현할 경우 이러한 방법은 병렬적으로 처리가 불가능하기 때문에, 기존에 병렬화된 pipeline 구조를 유지할 수 없게 된다. 본 연구에서는 하드웨어에 적용 가능한 keypoint selection 에 대한 연구를 제안하였다.

2.3.2 Clustering 기반의 feature matching

SIFT 에서 생성한 local patch descriptor 의 유사성만을 이용하여, correspondence 를 찾으면, 부분적으로 유사한 다른 부분의 descriptor 로 correspondence 가 이루어질 수 있다. 따라서, 초기의 correspondence 로부터 inlier 와 outlier 를 구분하는 방법들에 대한 연구들이 진행되고 있다.

이전 논문[9] 에서는 이미지를 rotation, translation, scale 의 변화만을 보이는 rigid scene 으로 가정하고, RANSAC(RANdom Sample Consensus) 방법을[9][11] 사용하여, 한 이미지에 point 들이 다른 이미지에서 어떤 위치에 대응하는지에 대한 affine transform 을 추정하여, affine transform 에 부합하는 correspondence 를 inlier 로 선택하고, 나머지를 outlier 로 선택하는 방법을 사용한다. 그러나 이와 같은 방법은 non-rigid image 의 변형이나, affine transform 에 의해서 표현되지 않는 complicated scene 같은 경우에는 효과적이지 못하게 된다. 따라서 최근 들어 non-rigid image 변화에 적합한 matching 알고리즘들이 연구되고 있다[31][32][33].

Non-rigid 한 이미지에서의 matching 을 위해 feature 간 geometric 한 정보를 사용하는 방법들이 있다. 이는 이미지 내의 keypoint 간 distance 나 angle 이 상대적으로 불변함을 이용하여, keypoint 들로 구성된 graph 를 사용하여 pairwise 한 대응 쌍을 찾거나, triples of points set 을 찾는 방법을 사용한다. 이러한 복수개의 keypoint 들을 사용해서 수행되는 high-order matching 은 local feature 의 유사성뿐만 아니라 keypoint 간의 위치 관계를 사용하여 matching 을 수행한다. 그러나 이러한 방법은 많은 수의 correspondence 가 존재하는 경우, outlier 의 조합에 의해서도 distance 나

angle 간의 불변한 성질을 만족하기 때문에, 잘못된 결과를 생성할 가능성이 많게 된다. 또한 이러한 방법은 연산의 복잡도가 높은 문제가 있다. 이러한 문제점을 개선하기 위해서 최근에 feature 간의 clustering 을 이용하여 신뢰성 높은 feature set 을 얻는 방법들이 있었다. 이와 같은 clustering 방법들은 비교적 정확한 결과를 제공하지만, 이미지 내의 모든 feature correspondence 를 대상으로 반복적으로 clustering 과정을 수행하기 때문에, correspondence 의 수가 늘어나면 기하급수적으로 수행 시간이 증가하게 된다[35][38].

본 연구에서는 이와 같이 clustering 에 의한 feature matching 방법에서 발생하는 연산의 복잡도를 개선하기 위해 region-constrained clustering 방법을 제안한다.

2.3.3 Watershed Transform

많은 응용 분야에서 실시간 이미지 segmentation 을 필요로 하고 있으며, 여러 가지 이미지 segmentation 알고리즘 가운데, watershed 알고리즘[39]은 불분명한 경계를 가지는 object 를 효과적으로 구분할 수 있는 방법이다[40]. 그러나 watershed 알고리즘은 전체 이미지에 대하여 gradient 를 기준으로 sorting 을 하고 이 순서에 따라 픽셀별로 주변값과의 관계를 조사하여 순차적으로 연산을 하기 때문에, 연산량이 많고 병렬화가 어렵다. 따라서, 입력 이미지의 크기가 커짐에 따라 watershed 알고리즘의 수행 속도를 급격히 저하시키게 되고, 그에 따라 실시간 처리가 어렵게 된다[43][44]. 이와 같은 segmentation 은 개별 결과뿐만 아니라, 시스템에서 전처리 과정이나

정보를 제공하는 단계로도 많이 사용되기 때문에, 빠른 연산이 필요하다.

이전 연구에서는 이미지의 변화된 부분만을 재계산하여, segment 이미지를 생성하는 방법이 있다. 이러한 방법은 이전에 계산된 부분과, 새로 계산해야 하는 부분을 설정하는데, 새로 계산하는 부분이 블록의 집합군의 형태로 나타나게 된다. 이 방법은 새로 계산하는 블록과 기존 계산된 블록과의 경계면에서 전체 이미지의 segmentation 과 다른 over-segment 된 결과를 생성한다. 또한 블록의 형태가 임의의 크기로 구성된 블록의 집합군의 형태를 가지므로 하드웨어 구현에 적합하지 않다[42]. 본 연구에서는 독립적으로 동일한 형태의 블록 단위로 수행이 가능한 블록 기반의 watershed 방법을 제안하고, 블록 경계에서 발생하는 부정확한 결과를 향상시키는 방법을 제안하였다.

제3장 Adaptive keypoint generation 하드웨어 구현

이번 장에서는 adaptive keypoint generation 을 위한 SIFT 하드웨어 구조에 대해 제안한다. 이를 위해서, 하드웨어 구조에 적합한 keypoint 조절 방법에 대해서 살펴보고, 제안된 하드웨어 구조에 대해서 설명한다.

3.1 SIFT 하드웨어 속도 향상을 위한 요소

SIFT 는 다른 여러 가지 local feature 방법 중 다양한 이미지 변형에 강인한 특성을 보이기 때문에 가장 널리 사용되고 있다[13][15][20]. 하지만 SIFT 는 많은 연산을 요구할 뿐만 아니라, 많은 양의 메모리를 필요로 하기 때문에, 실시간으로 구현하는데 어려움이 있다. 따라서 SIFT 의 연산을 간략화 해서, 연산 속도를 높이기 위한 연구들이 있어 왔다[14][19]. 또한 FPGA 같은 하드웨어 가속기를 사용하여 연산 속도를 높이는 연구들도 이루어져 왔다[16][17][18]. 이러한 하드웨어를 사용한 접근 방법들은 연산의 병렬처리나 메모리 재사용 방법 등을 통해 속도와 효율성을 높이는 방향으로 연구가 진행되어 왔다. 그럼에도 불구하여, 여전히 실시간 구현에 어려움을 가지고 있다. 본 연구에서는 하드웨어로 구현된 SIFT 에 적합한 속도 향상 방법을 제안한다.

이를 위해 먼저 SIFT 연산에 대한 블록별 특성을 살펴보면 그림 3.1 과 같이 연산량에 영향을 미치는 요소들이 서로 다름을 알 수 있다. Keypoint

detection 은 전체 연산량이 입력 이미지의 크기와 관련이 있는 부분이다. 따라서 입력 이미지의 크기만 관련이 있고, 이미지내에 존재하는 특성과는 관련이 없는 부분이다. 반면 descriptor generation 부분과 feature matching 부분은 전체 연산량이 keypoint detector 에서 발생한 keypoint 의 수와 관련된 부분으로 입력 이미지의 크기와는 관련이 없는 부분이다.

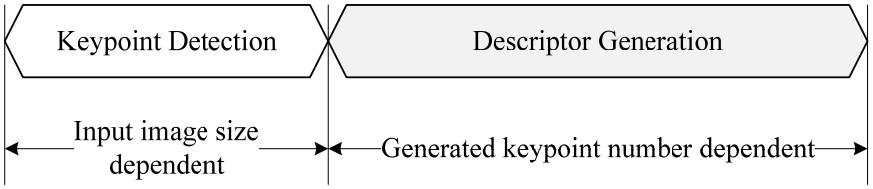


그림 3.1 블록별 연산량에 영향을 미치는 요인

하드웨어 구현된 SIFT 알고리즘에서는 keypoint detection 이 다 끝나기 전에 생성된 keypoint 를 사용하여, descriptor generation 부분을 시작할 수 있다. 따라서, 그림 3.2 와 같이 병렬 처리하도록 구성되어 있다. 그럼에도 descriptor 생성 부분은, keypoint detection 부분에 비해 많은 양의 연산을 필요로 하므로, keypoint detection 이 완료된 이후에도 descriptor generation 모듈을 수행하기 위한 상당한 시간이 필요하게 된다.

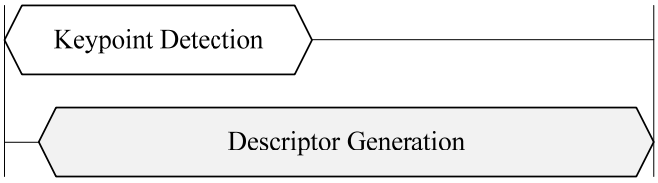


그림 3.2 병렬화된 keypoint detection 과 descriptor generation

앞에서 살펴본 바와 같이 keypoint 의 수는 전체적인 속도에 영향을 미치는 요소이기 때문에 keypoint detection 단계에서 keypoint 수를 줄임으로써, 전체적인 속도 향상을 하는 연구들이 있어 왔다. 그러나 단순히 전체 개수만 줄이는 방법을 적용할 경우에는 이미지내에 특징을 기술할 keypoint 가 특정 영역에만 존재하기 때문에, 이미지의 특성을 고려하여, keypoint 들이 고르게 분포할 필요가 있다.

이전 연구 방법들은 keypoint 수에 의해 발생하는 연산의 복잡도 문제를 해결하기 위해서, keypoint 의 생성이 완료된 이후에, 생성된 keypoint 들을 sampling 하는 방법을 사용하였다. 따라서, 그림 3.3 와 같이 구성된 병렬화된 SIFT 하드웨어 구조에 적용하기 어려운 문제가 있다.

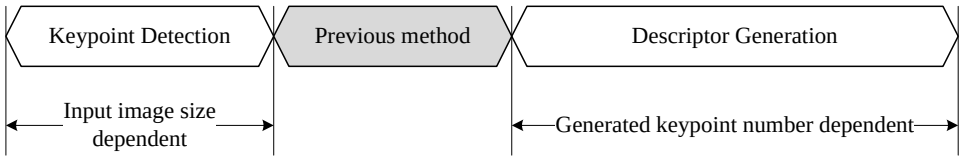


그림 3.3 이전 연구 방법

이와 같은 문제점 때문에, 하드웨어에 적합한 keypoint 를 조절할 수 있는 방법이 필요하다.

3.2 Hardware 에 적합한 keypoint 조절 방법

이전의 연구 방법들이 keypoint 생성 이후에 keypoint 조절하기 때문에, 병렬화된 하드웨어에 적용하기 어려운 문제점이 있었다. 따라서, 하드웨어에 적용하기 위해서는 keypoint 생성 단계에서 keypoint 수를 조절하는 방법이 필요하다. SIFT 는 DoG 이미지의 local extrema 로부터 keypoint localization 단계를 거쳐서 최종 keypoint 를 생성하게 된다. Localization 단계에서는 keypoint 주변이 low contrast 를 가지는 keypoint 를 제거하게 되는데, 이 과정은 최종 keypoint 수와 관련이 있게 된다. 이전 논문에서는 [9] $|D(\hat{x})| < 0.03$ 조건을 전체 이미지에 적용하여, 최종 keypoint 로 선택하였다.

Contrast threshold 값에 따른 keypoint 발생 수를 살펴보면, threshold 변화 비율에 비례하여, keypoint 수가 달라지는 것을 알 수 있다. SIFT 의 경우엔 각각 octave 별로 keypoint 가 생성되는데, octave 별로도 같은 경향성을 보임을 알 수 있다.

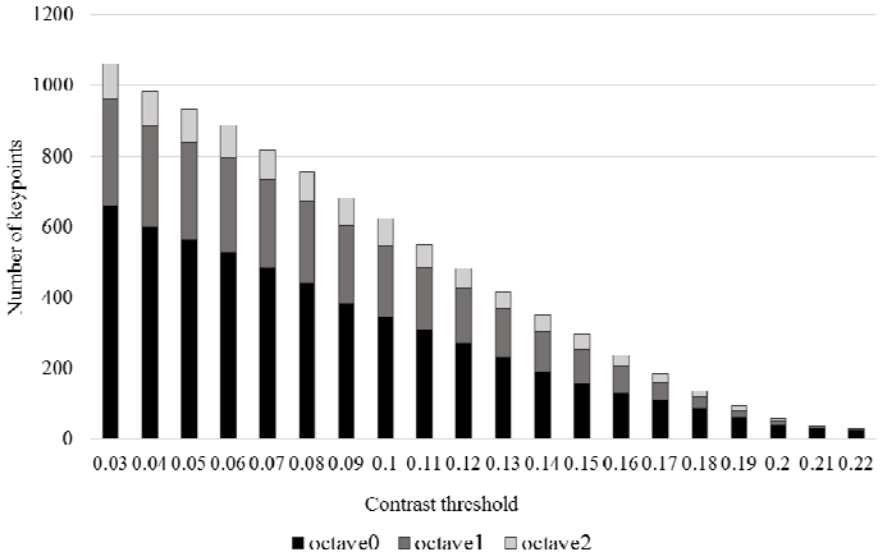


그림 3.4 Contrast threshold 에 따른 keypoint 수 변화

이와 같이 contrast threshold 는 keypoint 를 수를 조절할 수 있는 요소로 작용하고, keypoint detection 동작에 적용이 되어, 하드웨어로 구현된 SIFT 에서 keypoint 조절을 위한 방법으로 사용 가능 하다. 그러나 contrast threshold 를 통해 keypoint 의 개수는 조절이 가능하지만, 이전 연구의 문제점에서도 살펴본 바와 같이 keypoint 분포를 고려하면, keypoint 의 수를 줄이기 위해서, 이미지에 동일한 contrast threshold 값을 적용하는 것은, contrast 가 강한 영역의 주변의 keypoint 만 남게 되고, low contrast 영역 주변의 keypoint 는 없어지게 된다. 따라서, 이미지를 표현할 feature 들이 일부의 영역에만 존재하게 된다. 따라서 matching이 될 가능성이 있는 전체 local patch 가 줄어들게 된다. 이러한 문제점 때문에, keypoint 의 수를 줄이는 문제에서는, keypoint 의 분포도 함께 고려가 되어야 한다.

이와 같이 상대적인 위치 분포를 유지하면서, keypoint 를 줄이기 위해서는 특정 영역에 많은 수의 keypoint 가 발생하는 영역에 대해서만 keypoint 를 조절하는 방법이 필요하다. 그러나 SIFT 의 keypoint 를 생성하기 전에는 특정영역에 keypoint 가 많이 발생하는지 여부를 알 수 없기 때문에 어떤 영역의 keypoint 를 줄여야 할지에 대한 정보도 알 수 없게 된다. 따라서 SIFT keypoint 생성 이전에 상대적으로 SIFT keypoint 가 많이 발생할 부분을 예측하면, 영역별로 다른 contrast threshold 값을 통해서 low contrast 주변의 keypoint 는 유지하면서, high contrast 주변에 많이 발생하는 keypoint 만 조절을 할 수 있게 된다.

3.3 FAST detector 를 통한 이미지 특성 예측

SIFT 는 DoG(Difference Of Gaussian)이미지를 사용하여, DoG 의 local maxima 위치에서 keypoint 를 생성한다. 이러한 DoG 는 edge 나 blob 주변에서 강하게 반응하기 때문에, edge나 blob 주변에서 SIFT의 keypoint 들이 많이 나타나게 된다. SIFT 에서는 이와 같은 keypoint 를 생성하기 위해서, scale 단계에 따라서, 여러 번의 Gaussian filtering 과정을 수행하게 된다.

다른 feature detector 들은 다른 과정을 통해 이미지에서 특징점들을 생성한다. 이와 같은 feature detector 들이 생성하는 keypoint 의 위치들은 서로 일치하지 않지만, 발생된 keypoint 의 분포들에는 서로 상관성을 가지고 있다.

예를 들어 단일 색상의 배경 영역 부분에서는 keypoint 의 발생 분포가 낮고, 건물과 같은 많은 수의 edge 와 corner 가 존재하는 영역에서는 keypoint 의 발생 분포가 높게 생성된다. 이는 feature 가 물체의 distinctive 한 특징을 나타내기 때문에, 주변에 이미지 변화가 없는 영역보다는 이미지 변화가 많은 영역에 많이 나타나게 된다. 따라서 일정 영역 기준으로 보면, 다른 종류의 feature detector 에서 생성된 keypoint 간에는 keypoint 의 위치를 다르지만, 영역내의 발생 분포 밀도 측면에서는 상관성을 가지고 있다. 따라서 SIFT 에서 발생하는 keypoint 발생 밀도를 예측하는데, 다른 종류의 detector 를 결과를 활용할 수 있다. 특히 corner 는 서로 다른 방향성을 가진 edge 의 교차점에서 관찰되는데, 일정 범위 내에서 살펴보면, 결국 여러 개의 edge 로 구성된 영역 주변에서 SIFT 의 keypoint 가 많이 발생하게 된다.

본 연구에서는 SIFT 의 keypoint 의 발생 분포를 예측하기 위해서, corner detector 의 한 방법인 FAST(Features from Accelerated Segment Test)[22] 를 사용하였다. 또한 FAST detector 는 빠르게 keypoint 를 생성할 수 있기 때문에, SIFT 의 keypoint 의 분포를 사전에 예측하는 방법으로 효과적으로 사용 가능하다.

3.3.1 SIFT keypoint 와 FAST keypoint 분포의 상관성

본 절에서는 블록 단위로 발생하는 SIFT keypoint 와 FAST keypoint 분포 간의 상관성을 설명한다. FAST detector 는 현재 픽셀 위치에서 3 픽셀 떨어진 위치의 픽셀들을 원 형태로 조사한다. 반면 SIFT 의 경우 edge 주변이나 blob 의 center 위치에서 keypoint 가 생성되게 된다. 이때, 발생 위치는 Gaussian kernel size 에 따라서 달라지게 된다. 따라서 두 detector 간에 발생하는 위치는 서로 다르게 된다. 그러나 블록별 분포간에는 연관성을 가지고 있다.

먼저 두 종류의 특성이 다른 이미지들에 대해서 SIFT 와 FAST 에 의해서 발생하는 keypoint 에 대한 분포를 도출하고, 분석을 진행하였다. 그림 3.5 의 자연 영상 이미지와 그림 3.6 의 인공 구조물에 대한 이미지들을 가지고 FAST 와 SIFT 에서 생성된 keypoint 들간에 블록단위 분포의 상관성을 실험하였다. 각각 종류의 이미지들은 500 개의 dataset 으로 구성하였다.

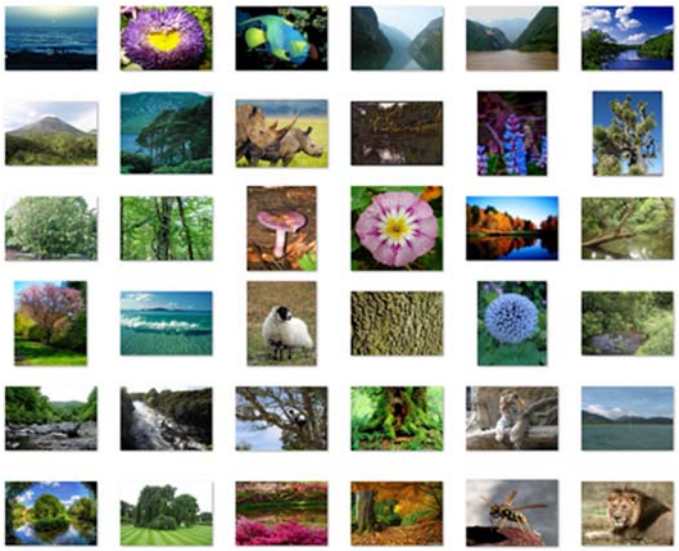


그림 3.5 자연영상 이미지 예

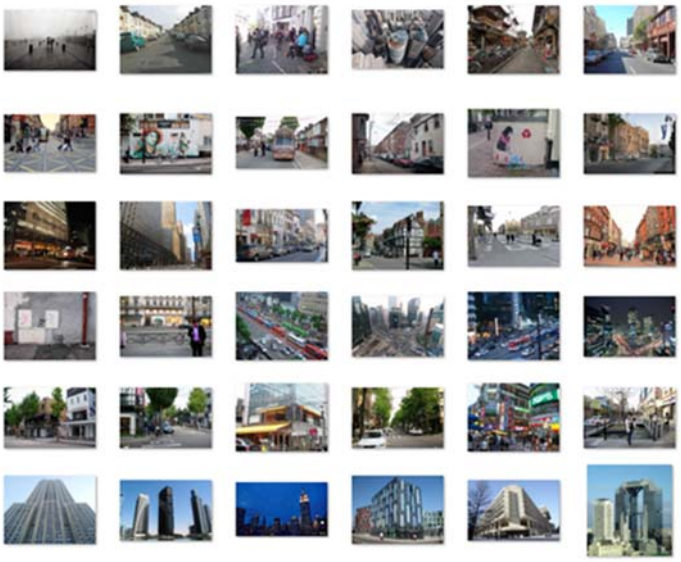


그림 3.6 인공 구조물 이미지 예

표 3.1 FAST 와 SIFT 에 의한 블록별 keypoint 분포의 상관성

		FAST 에 의한 블록내 이미지의 복잡도	
		높다	낮다
SIFT 에 의한 블록내 이미지의 복잡도	높다	I	II
	낮다	III	IV

블록별로 FAST keypoint 분포에 따라서 SIFT 의 keypoint를 예측하는 경우엔 표 3.1 과 같이 4가지의 경우가 생기게 된다. Case I 은 FAST keypoint 의 갯수에 의해서 블록내 이미지가 복잡하다고 판단된 영역에 실제로 SIFT 에 의해서도 keypoint 가 많이 나오는 경우로 SIFT 에 의해서도 복잡도가 높게 판단된 경우이다. Case II 은 FAST 에 의해서 블록내 이미지의 복잡도는 낮게 판단되었으나, SIFT 에 의해서는 복잡도가 높게 나오는 경우이다. Case III 는 FAST 에 의해서는 복잡도가 높은 영역이 SIFT 에 의해서는 복잡도가 낮게 판단된 경우이며, Case IV 는 FAST 와 SIFT 에 의해 모두 블록내 이미지의 복잡도가 낮은 영역으로 판단된 경우이다.

따라서, Case I, IV 는 맞게 판단된 블록으로 SIFT 에 의해서 발생하는 keypoint 가 맞게 조절되는 경우이다. 반면, Case II 는 FAST 에 의해서 복잡도가 낮은 영역으로 판단이 되었지만, 실제로는 SIFT 에 의해서는 keypoint 가 많이 발생하는 영역으로 keypoint 가 개수가 줄어야 하지만, 줄어들지 못하는 영역이다. 반대로 Case III 는 FAST 에 의해서 복잡도가 높은 영역으로 판단하고, SIFT 의 keypoint 의 수를 줄이지만, 실제로는 SIFT 의 keypoint 발생 분포가 낮은 영역으로 영역을 기술할 keypoint 가 사라질

수 있는 영역이다.

실제 정확도는 전체 블록 중 Case I, IV 로 판단하는 블록은 맞게 판단한 부분으로 하고, Case III 는 과도하게 keypoint 줄이는 영역, Case II 는 keypoint 를 줄일 필요가 있지만, 줄이지 못하는 영역으로, Case II, III 가 상관성을 정확하게 예측하지 못한 부분이 된다.

예를 들어 그림 3.7 (a) 의 원본 이미지에 대해서 (b)는 FAST 의 keypoint 수에 따라서 블록의 복잡도를 분류한 결과이고, (c) 는 SIFT 에 의해서 블록의 복잡도를 분류한 결과이다. 좌상단이 복잡도가 낮은 블록들이고, 우하단으로 갈수록 이미지의 복잡도가 높은 블록이다. 그림 3.7(b), (c) 는 대체적으로 비슷한 경향성을 보이지만, 블록간 정확히 분류가 일치하지 않게 된다.

표 3.2 는 실험 영상에서 이러한 차이를 설명한다. 실험은 FAST 에서 블록에서 발생하는 평균적인 keypoint 의 수보다 30% 많이 발생하는 블록을 복잡도가 높은 블록으로 선정하였다. 두 종류의 테스트 이미지들에 대해서, 맞게 예측한 경우(Case I, IV) 는 자연영상 이미지의 경우 77%, 인공 구조물 이미지의 경우 79% 정도로 비슷한 상관관계 예측결과 수치를 보여준다.

표 3.2 FAST 와 SIFT 에 의한 블록별 상관관계 예측 결과

	Case I, IV	Case II	Case III
자연영상	77%	11.5%	11.5%
인공구조물영상	79.1%	10.5%	10.5%

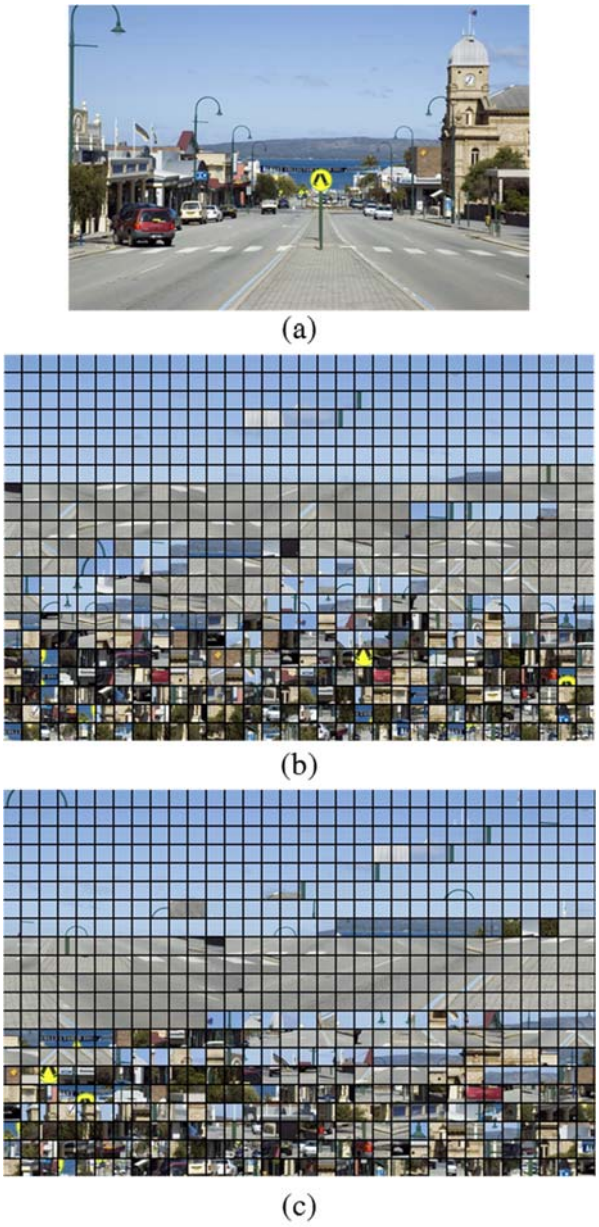


그림 3.7 블록별 복잡도에 따른 분류 (a) 원본 영상 (b) FAST 에 의한 분류 (c) SIFT 에 의한 분류

3.4 Adaptive keypoint generation 하드웨어 구조

전체 SIFT 하드웨어는 그림 3.8 와 같이 크게 4 부분의 기능별 모듈로 구성되어 있다. Gaussian filter bank 에서는 외부 메모리에 저장된 영상 데이터를 Gaussian filter bank 내부 source line buffer 에 저장하고, 저장된 이미지는 모든 Gaussian filter 에 동시에 전달되어 다양한 Gaussian filter kernel 에 대해서 filtering 이 동시에 이루어지게 된다. Filtering 된 이미지로부터 생성된 DoG 이미지는 keypoint detector 로 전달되며, 또한 생성된 scale-space 정보는 descriptor 를 생성할 때 이용하기 위해서 memory 에 저장된다. Keypoint detector 모듈에서는 DoG 이미지로부터 local extrema 를 찾고, contrast check 단계와 edge 근처에 있는 값을 제거하여, 최종적인 keypoint 값을 생성하게 된다. Gradient generation 모듈은 keypoint 주변의 local image 영역에 대해서 gradient 값을 생성하게 되고, 생성된 gradient 값의 histogram 을 사용하여, descriptor 를 생성하게 된다.

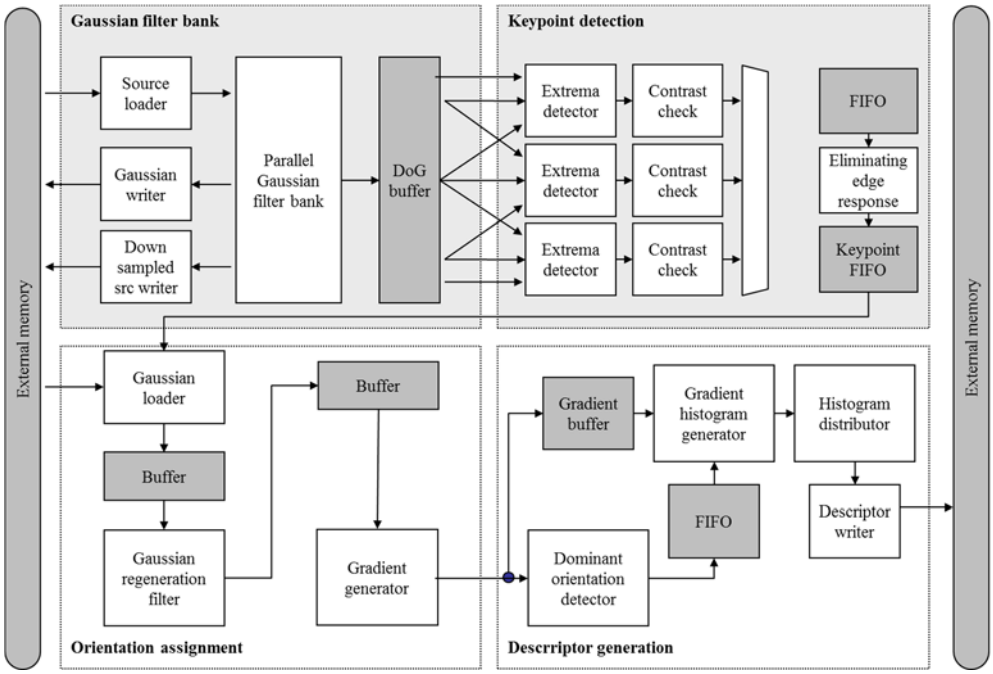


그림 3.8 SIFT Hardware block diagram

이후 절에서는 전체 하드웨어 구조에서 adaptive keypoint 생성을 위한 hardware 구조를 제안한다.

3.4.1 Gaussian filter bank 구조

본 절에서는 기존 SIFT 의 Gaussian filter bank 구조를 설명하고, adaptive keypoint generation 을 위해 제안하는 하드웨어와 기존 Gaussian filter bank 간에 연결 동작에 대해 설명한다.

Gaussian filter bank 는 외부 메모리에 저장된 입력 이미지로부터 Gaussian filtering 된 이미지를 생성하고, 이를 통해 DoG 이미지를 생성하는 부분이다. Gaussian filtering 을 수행하기 위해서, 사용하는 내부 메모리의

양을 줄이기 위해서, Gaussian filter bank 는 블록 단위로 연산이 진행된다.

전체 이미지를 이미지의 폭을 기준으로 96 픽셀 만큼씩 처리하고, 다음 블록을 처리하도록 되어 있다. 그림 3.9은 블록 단위 처리를 위한 연산 순서를 보여준다.

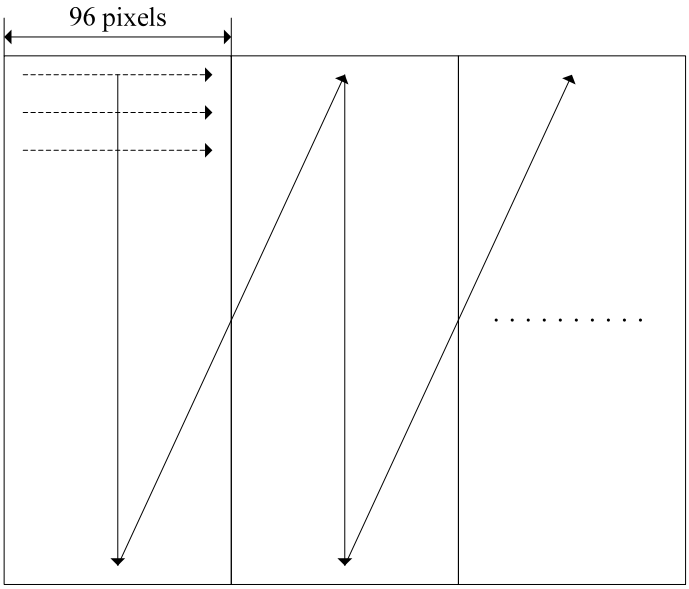


그림 3.9 입력 이미지 data loading 방식

또한 (96 x image height) 의 데이터를 처리하기 위해서 데이터를 한번에 가져오지 않고, circular buffer 형태의 block memory 를 사용하여, 순차적으로 처리하도록 되어 있다. Block memory 의 크기는 그림 3.10와 같이 size 가 128(32 word)이고, 전체의 개수가 64 개인 buffer 를 사용하여 저장한다.

Source loader 로부터 들어오는 픽셀들은 buffer 에 차례로 저장된다. 128 픽셀의 데이터가 다 저장된 이후에는 다음 line 의 128 개 데이터가 buffer 에 저장된다. Source buffer 는 circular 하게 동작하게 된다. 따라서 64 개의

buffer 가 다 채워지면, 가장 먼저 저장된 buffer 는 제일 아래로 내려가고 새로운 128개의 data 가 buffer 에 저장되게 된다.

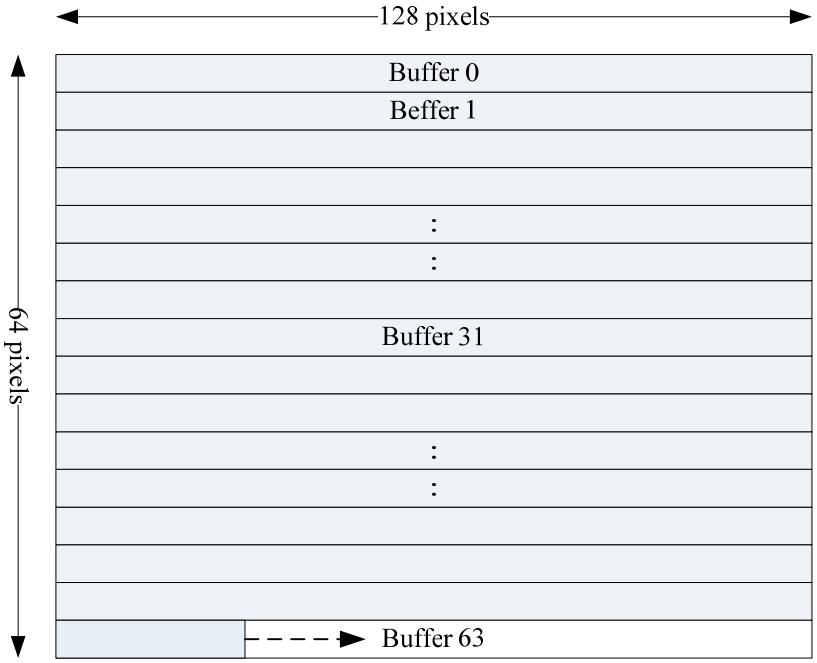


그림 3.10 Source buffer 구조

내부에서 사용하는 circular buffer 의 메모리 크기는 Gaussian filter 의 kernel size 와 관련이 있다. Gaussian filter 는 Gaussian 함수를 FIR(Finite Impulse Response) filter 로 근사화하여 계산한다. Gaussian 함수는 중심점에서 멀어질수록 함수값이 급격히 작아지므로 실용적으로 사용시에는 일정 범위의 filter 계수만을 적용하여 사용하게 된다. 본 연구에서는 함수값이 Gaussian function 최대값의 1% 이하로 떨어지는 지점까지의 함수값을 필터 계수로 사용하였다. 이 지점은 $2\sigma\sqrt{ln10} \approx 3\sigma$ 에 위치하게 된다. Gaussian

함수는 좌우 대칭을 이루기 때문에, Gaussian kernel 의 크기는 $Size_{guassian} = 2 \cdot Round(3\sigma) + 1$ 로 계산된다[4]. 본 연구에서는 $\sigma_0 = 1.6$ 의 값을 사용하였다. 따라서 scale-space 를 구성하기 위해서, filter scale 에 따른 Gaussian kernel 크기는 표 3.3 와 같이 만들어진다.

표 3.3 Filter scale 에 따른 Gaussian kernel 크기

Filter scale	σ_0	$\sigma_0 \cdot 2^{1/3}$	$\sigma_0 \cdot 2^{2/3}$	$\sigma_0 \cdot 2$	$\sigma_0 \cdot 2^{4/3}$	$\sigma_0 \cdot 2^{5/3}$
Gaussian Kernel size	11	13	17	21	25	31

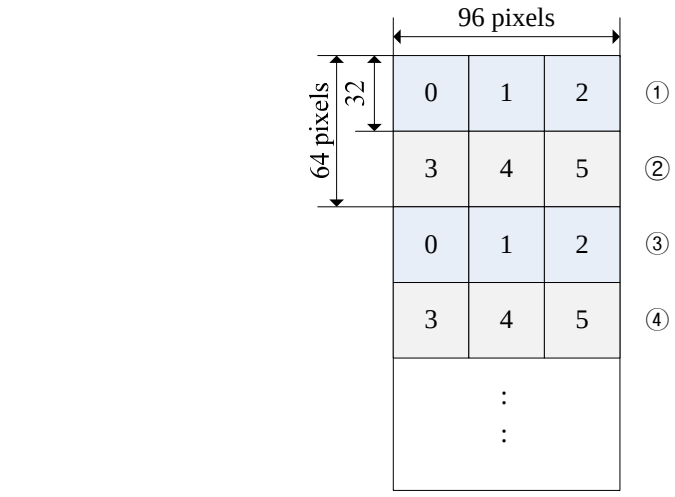
Gaussian filter bank module 은 가로 방향과 세로 방향에 대해서 각각 Gaussian filtering 을 수행한다. 따라서 source line buffer 의 크기는 SIFT 동작을 위해서, 가장 큰 Gaussian kernel 의 Size 를 기준으로 최소 31 line 이상의 Data 가 필요하게 된다. 따라서 31 개의 line buffer 가 다 채워지게 되면, 세로로 한 픽셀을 읽어 Gaussian filter bank 에 보내게 된다.

또한 가장 큰 kernel size 를 기준으로 96 개의 픽셀을 filtering 을 하기 위해서 위해서 좌, 우 15 픽셀의 data 가 추가로 필요하고, data 를 word 단위로 처리하기 때문에, 좌, 우 16 픽셀에 해당하는 크기만큼의 data 를 추가로 저장하게 된다. 따라서 한 line buffer 의 크기는 128 로 설정한다.

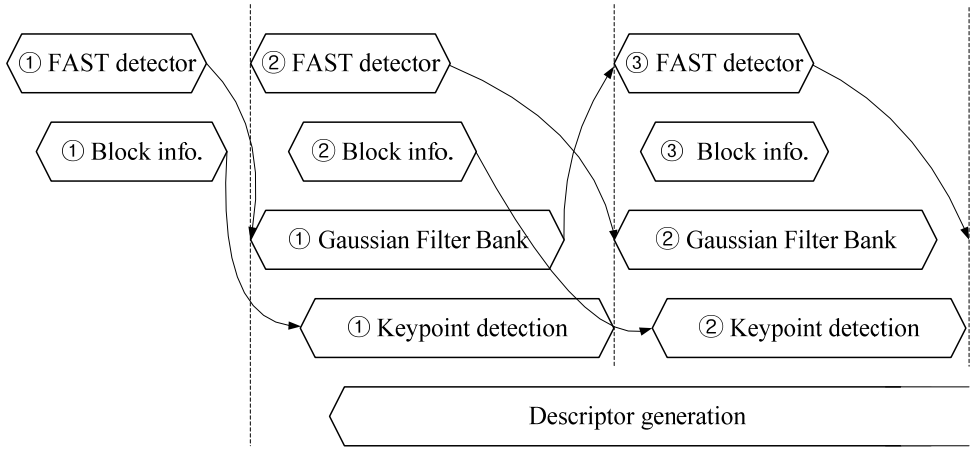
제안한 hardware 구조에서는 해당 픽셀의 SIFT 의 연산 이전에 keypoint 의 일정 영역 내의 발생 분포를 예측하기 위해서, 추가적으로 32 line 의 data 를 source line buffer 에 저장하여 사용하였다.

본 연구에서는 32x32 단위를 keypoint 분포를 예측하기 위한 단위로 사용하였다. SIFT 는 (96 x image height) 단위로 나누어 연산이 진행되므로 그림 3.11 (a) 와 같이 source buffer 에는 6 개의 keypoint 를 예측하기 위한 단위가 만들어지게 된다. 따라서, 세로 방향의 Gaussian filtering 을 위해서 필요한 buffer 의 line 수는 32 이지만 미리 keypoint 분포를 예측하기 위해서, 그림 3.10 에서와 같이 circular buffer 구조의 크기를 64 line 으로 구성하였다.

Keypoint 분포를 예측하기 위한 FAST detector 와 Gaussian filter bank 는 그림 3.11 (b) 와 같이 시간적으로 FAST detector 가 먼저 수행되어 ① 영역에 대한 keypoint 분포를 예측한 이후에 그림 3.10(b) 와 같이 ① 영역의 Gaussian filtering 을 수행하도록 되어 있다. FAST detector 는 Gaussian filter bank 의 수행시간보다 빠르게 수행되기 때문에, source line buffer 에서 data 를 읽어오는 기준은 SIFT 의 Gaussian filtering 의 연산속도에 영향을 받게 된다. 또한 FAST detector 와 Gaussian filter bank 는 handshaking 구조로 동작하기 때문에, 그림에서와 같이 FAST 는 ② 번 영역에 대한 FAST 의 연산이 완료되면, Gaussian filter bank 에 이를 알려주고, ③ 번 영역의 계산을 수행하고, Gaussian filter bank 는 ② 번 영역에 대한 Gaussian filtering 을 수행하게 된다. 따라서 6 개의 블록 정보를 저장할 공간만 있으면, 순차적으로 사용이 가능하다.



(a)



(b)

그림 3.11 FAST 와 Gaussian filter bank 연동 구조 (a) Keypoint 예측 블록 구성 (b) Timing diagram

위와 같은 buffer 구조를 통해 순차적으로 keypoint 분포 예측과 Gaussian filtering 이 가능하게 되며, 전체 수행시간 측면에서 SIFT 만 수행

하는 하드웨어 구조에서 초기 3개 블록에 대한 keypoint 발생을 예측하기 위한 FAST detector 의 연산에 의한 latency 만 가지게 된다.

3.4.2 FAST detector

블록내의 이미지 특성을 파악하기 위해서 본 연구에서는 FAST detector 에서 발생하는 keypoint 의 분포를 사용한다. 이를 위해 FAST hardware 의 동작에 대해서 설명한다. FAST detector 는 하나의 픽셀에 대한 keypoint 여부를 조사하기 위해서 7x7 윈도우 크기의 값을 필요로 한다. 또한 non-maximal suppression 을 통해 최종적으로 keypoint 를 판단하기 위해서 현재 keypoint 를 중심으로 3x3 범위내에서 발생된 keypoint 의 score 값을 사용한다.

따라서 한 픽셀을 FAST keypoint 로 최종 판단하기 위해선 입력 이미지 기준으로 9x9 의 주변 픽셀값을 필요로 한다. 따라서, 7x9 단위의 블록이 raster-scan order 로 데이터를 처리하게 된다. 7x9 윈도우를 사용하게 되면, non-maximal suppression 과정을 수행시에 필요한 위, 아래의 keypoint 에 대한 score 값을 가지고 있기 때문에, non-maximal suppression 을 동시에 수행 가능하다

7x9 윈도우는 그림 3.12 과 같이 3개의 위치에 대한 corner 여부를 동시에 조사할 수 있다. 점선에 해당하는 픽셀들은 ‘x’ 로 표시된 위치를 corner 로 판단하기 위해서 필요한 주변 픽셀들을 표시한다.

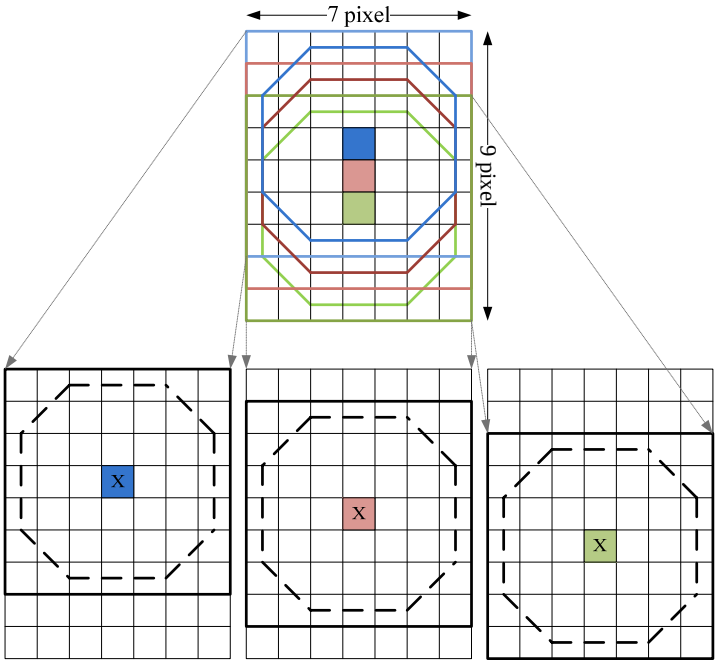


그림 3.12 7x9 window corner detection

따라서 FAST detector 를 위한 하드웨어 구성은 그림 3.13 와 같이 3 개의 corner detection 과 score 를 계산하는 부분과, non-maximal suppression 을 처리하는 부분으로 구성된다.

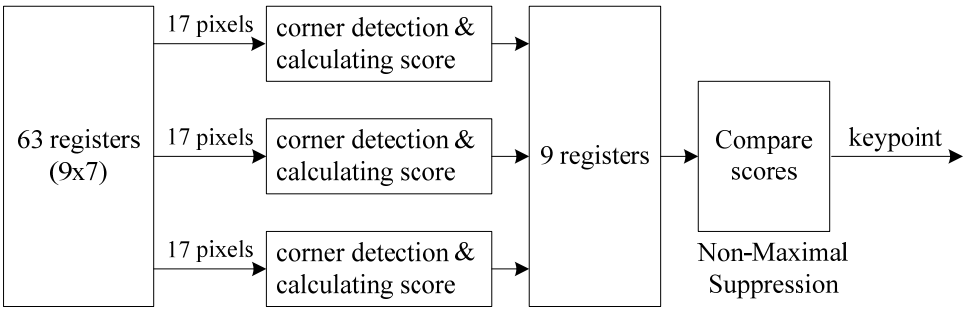


그림 3.13 FAST detector 하드웨어 구조

3.5 하드웨어 구조

그림 3.14 은 이전 절에서 설명한 Gaussian filter bank 구조와 FAST detector 를 적용한, adaptive keypoint generation 을 위한 하드웨어 구조를 보여준다.

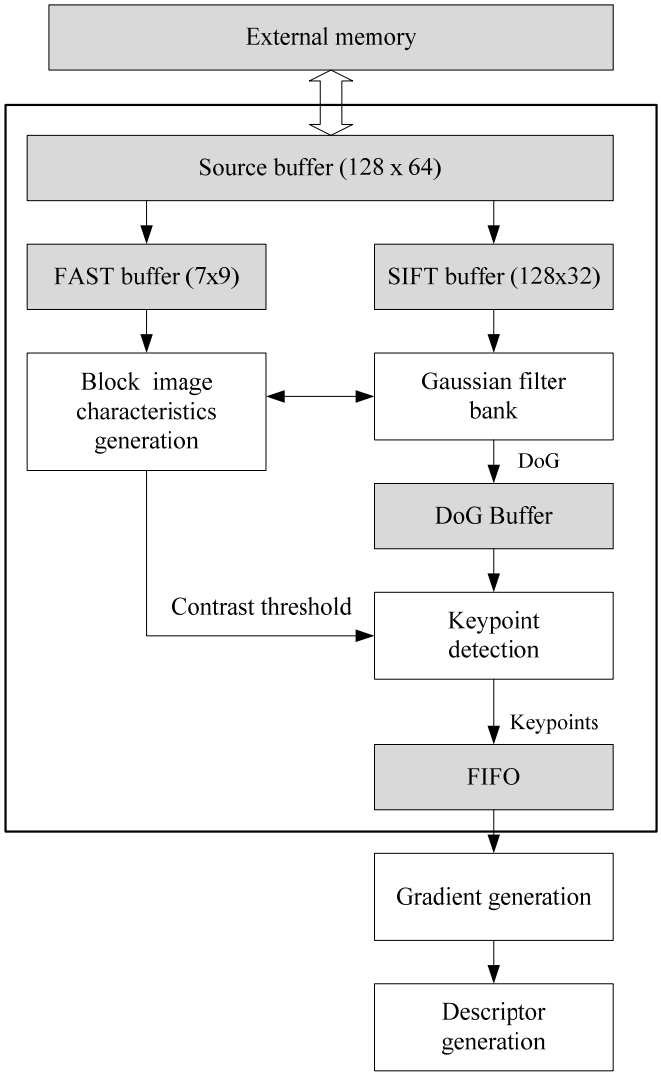


그림 3.14 Adaptive keypoint 생성을 위한 hardware 구조

전체 시스템은 외부 메모리에 저장된 데이터를 읽어서 source buffer 에 저장하는 ‘Source loader’ 부분, 저장된 source buffer 로 부터 블록내의 이미지의 characteristics 를 예측하는 ‘Block image characteristics generation’ 부분, keypoint 를 생성하기 위한 ‘Gaussian filter bank’, ‘Keypoint detection’ 부분, 생성된 keypoint 로부터 gradient 를 생성하는 ‘Gradient generation’ 부분과 ‘Descriptor generation’ 부분으로 이루어져 있다. 이중 adaptive keypoint generation 을 위해선 점선 부분으로 표시된 영역에 블록 이미지의 복잡도를 예측하는 부분을 추가하고, 이를 통해 keypoint 생성단계에서, keypoint 조절이 가능한 구조를 적용하였다.

3.5.1 Block image characteristics generation

그림 3.15 는 'Block image characteristics generation' 부분에 대한 하드웨어 구조를 보여 준다.

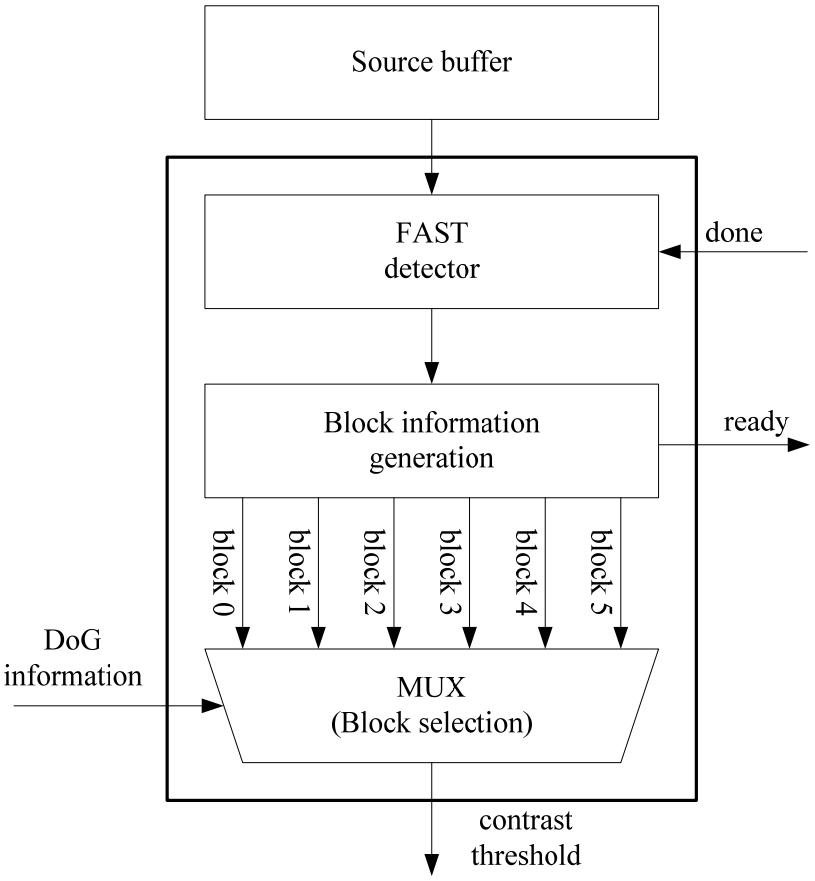


그림 3.15 Block image characteristics generation

'Block image characteristics generation' 부분은 입력으로 source loader 를 사용하여 저장한 source buffer 의 data 를 받고, 블록 단위의 이미지의 복잡도 정보 생성이 끝나는 시점에 'Gaussian filter bank' 에 해당 블록의

Gaussian filter bank 의 시작이 가능함을 알려 주도록 되어 있다. 또한 생성된 블록 단위의 이미지의 복잡도 정보를 사용하여, keypoint detector 모듈이 수행되는 부분에 맞는 contrast threshold 를 전달해 주도록 되어 있다. 블록 이미지의 복잡도가 생성되는 시점은 해당 위치의 contrast threshold 가 동작되는 시점과 다르게 되기 때문에, 이미 저장된 블록 단위의 정보로부터 keypoint detector 가 DoG data 를 읽어가는 순서에 맞는 contrast 값을 제공해 주는 과정이 필요하게 된다. 블록별 이미지의 복잡도 정보는 블록내에 발생한 FAST keypoint 수의 합을 사용하여 결정한다. 또한 이를 통해 표 3.4 와 같이 이미지의 복잡도를 판단하고 contrast threshold 값을 조정하게 된다. 이때의 threshold 값은 적용되는 application에 따라서 조절 가능하다. FAST keypoint 의 발생 분포가 높은 블록은 '복잡'으로 판단하고, 이 블록은 contrast threshold 의 값을 증가한다. 반대로 '단순'으로 판단된 영역에서는 contrast threshold 를 높일 경우 keypoint 가 없어질 가능성이 있으므로 contrast threshold 값을 유지한다.

표 3.4 블록별 이미지 복잡도에 따른 contrast threshold

	블록내 FAST keypoint 수	블록내 이미지 복잡도	Contrast threshold
I	0	-	-
II	# of FAST KP < th	단순	유지
III	# of FAST KP > th	복잡	증가

SIFT keypoint detector 단계에서 적용되는 contrast threshold 값은 SIFT keypoint detector 에 전달되는 DoG 이미지 위치에 따라서 달라지게 된다. 따라서, Block image characteristics generation 에서는 입력되는 DoG 위치 정보를 통해서 그에 맞는 블록 정보를 발생하도록 되어 있다.

3.5.2 Gaussian filter bank

그림 3.16 Gaussian filter bank 구조를 보여준다. Gaussian filter bank 는 입력 이미지로부터 DoG 이미지를 생성하는 부분이다. 이때, Gaussian filter bank 에서 사용하는 입력 이미지가 저장된 source buffer 는 앞 절의 'Block image characteristics generation' 에서 사용하는 공간과 동일한 공간이다. 3 개의 line buffer 는 각각 하나의 octave 에 대한 source image 의 line 을 저장한다. Octave 1 와 octave 2 에 대한 연산에서 사용하는 입력 이미지는 현재 octave 의 (S+1) 번째 Gaussian-blurred image $L_s(x,y)$ 을 1/2씩 down-sampling 하여 얻을 수 있다. Octave 1, Octave 2 를 위해 사용하는 down-sampling 된 이미지는 외부 메모리에 저장되어, 해당 Octave 계산시에 외부로부터 가져와서 사용하게 된다. 이와 같은 Gaussian filter bank 를 동작은 'Block image characteristics generation' 에서 생성되는 ready 신호에 따라서 동작하게 된다. 즉, Gaussian filtering 을 수행하는 픽셀에 대한 이미지의 복잡도 정보가 생성된 이후에 동작하도록 되어 있다.

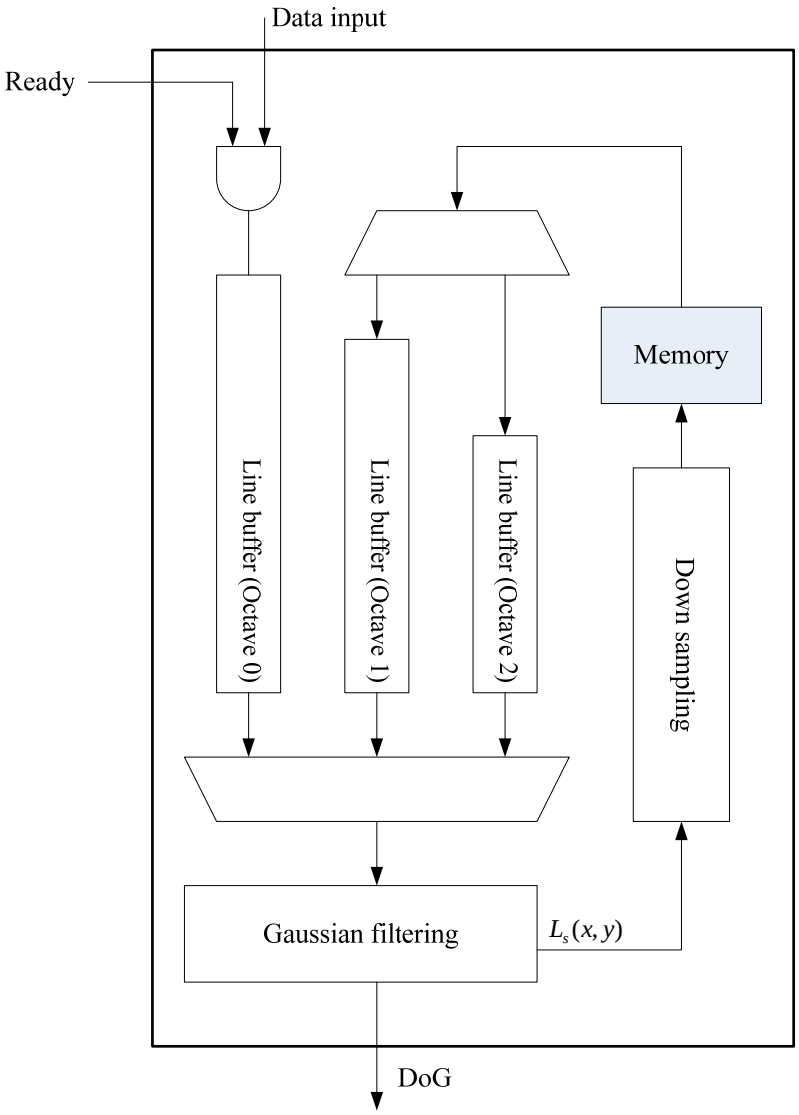


그림 3.16 Gaussian filter bank

3.5.3 Keypoint detector

그림 3.17 은 keypoint detection 을 위한 하드웨어 구조를 보여 준다. 본 연구에서는 5개의 DoG 이미지를 생성한다. 이를 통해 3 개의 extrema detector 를 위한 조합이 가능하게 된다. 속도를 향상하기 위해서, 하드웨어에서는 3개의 extrema detector 가 병렬적으로 동작하는 구조를 가지고 있다. 검출된 local extrema 는 contrast check 단계를 거치게 되는데, 이때, 'Block image characteristics generation' 의 contrast threshold 값이 사용된다. 이후에 이루어지는 'Eliminating edge response' 모듈은 검출된 keypoint 후보군에 대해서만 연산이 이루어지기 때문에, 많은 연산량을 필요로 하지 않는다. 또한 각각의 extrema detector 와 contrast check 에 의해서 생성되는 keypoint 는 서로 다른 개수가 생성된다. 따라서, Eliminating edge response 을 여러 개로 구성할 경우, load imbalance 하기 때문에, 병렬화로 인한 효과를 극대화 하기 어렵게 된다. 또한, 생성된 keypoint 는 descriptor 계산을 위해서 저장된다

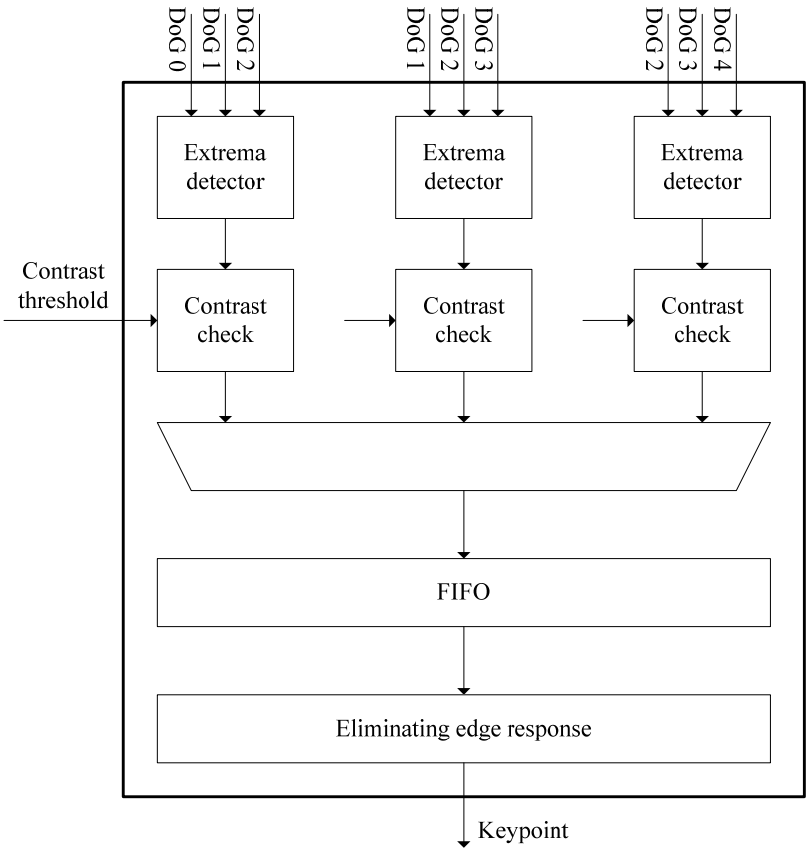


그림 3.17 Keypoint detection

3.6 성능 평가

이번 장에서 제안된 하드웨어 구조는 이미지에서 발생하는 keypoint 를 블록별로 adaptive 하게 조절할 수 있는 방법을 제공하고 있다.

제안된 방법은 이미지내에 SIFT keypoint 발생 분포가 높은 영역과 keypoint 발생 분포가 낮은 영역을 SIFT keypoint detector 동작 이전에 예측하여, SIFT 에서 발생하는 keypoint 의 수를 조절할 수 있도록 되어 있다. 또한 keypoint 발생 분포를 예측하기 위한 부분은 Gaussian filter bank 동작에 비해서 빠르게 수행 가능하여, 하드웨어로 구성된 SIFT 의 동작 구조에 영향을 미치지 않고 병렬로 수행이 가능하다.

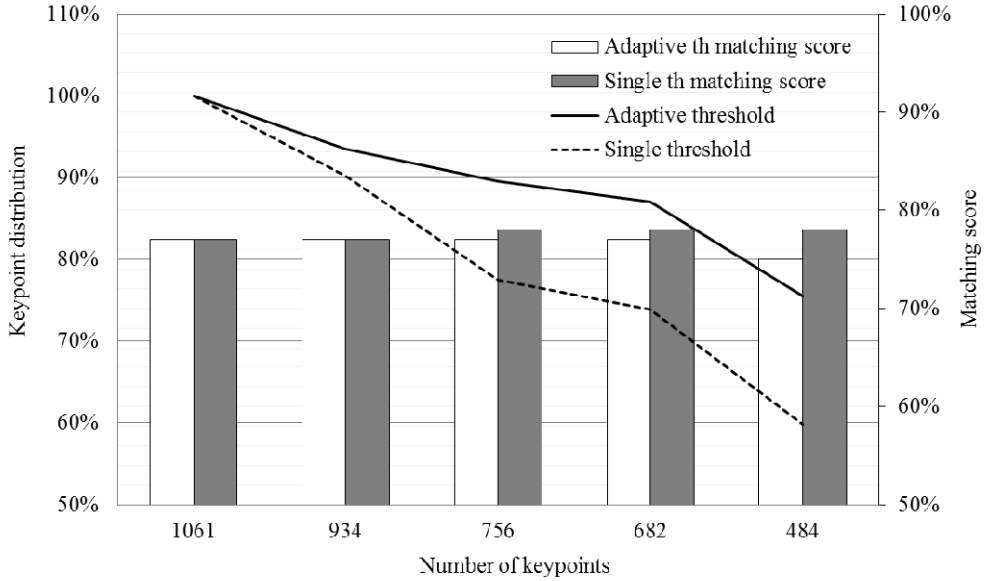
3.6.1 성능 실험 결과

그림 3.18 은 Graffiti 이미지를 사용해서 adaptive threshold control 방법에 의해 keypoint 수를 조절하였을 경우와 전체 이미지에 대해서 single threshold 를 적용하였을 경우에 대한 keypoint 분포 변화를 보여준다.

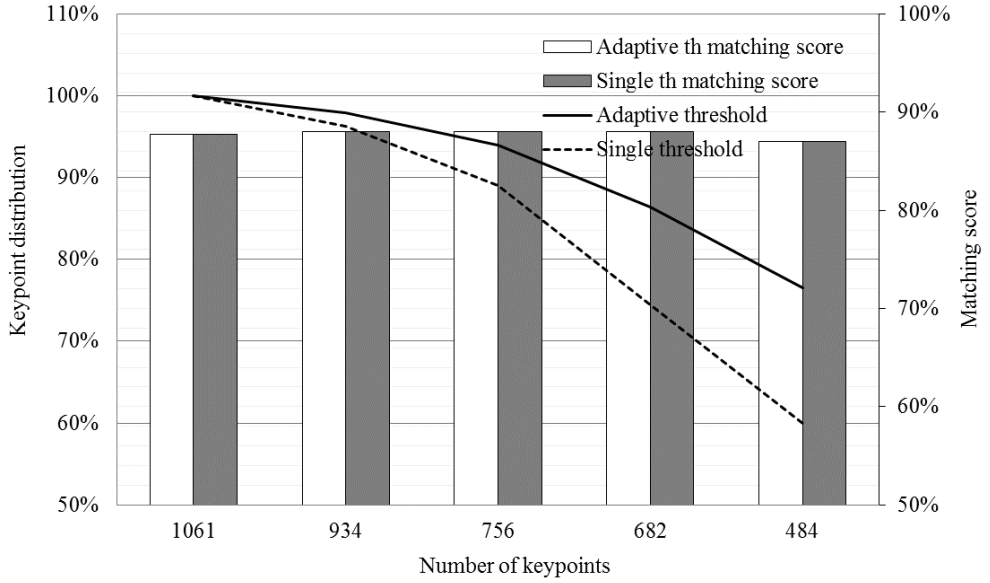
그림 3.18 의 꺾은선 그래프는 Lowe[9] 논문에서 사용한 0.03 contrast threshold 를 사용했을 때 발생하는 keypoint 분포를 기준으로 keypoint 수를 줄였을때의 분포를 상대적으로 표시한다. 또한 그림 3.18 의 막대형 그래프는 각각의 방법에서의 matching score 를 보여준다. 그림 3.18 에서 보이는 바와 같이 adaptive threshold 방법을 적용하였을 경우 keypoint 의 분포가 상대적으로 더 고르게 분포함을 알 수 있다.

그림 3.19 는 (b) 의 초기 SIFT keypoint 분포로부터 single threshold 값을 사용해서, keypoint 의 수를 줄였을 경우와 adaptive threshold 값을 적용

해서, keypoint 를 줄였을 경우에 대한 keypoint 분포를 보여준다. 그림 3.19 (c), (e) 는 초기의 keypoint 로부터 single threshold 값을 조절하여, 756, 484 개로 keypoint 수를 줄였을 경우에 대한 분포를 보여 주고, 그림 3.19 (d), (f) 는 adaptive threshold 를 적용하였을때의 결과를 보여준다. Adaptive threshold 를 적용하였을 경우에서 keypoint 의 분포가 더 넓게 분포 됨을 알 수 있다.



(a)



(b)

그림 3.18 keypoint 수에 따른 keypoint 분포 (a) Graffiti (b) Boat

표 3.5 는 여러 종류의 입력 이미지에 대해 keypoint 를 조절했을때 실험 결과를 보여준다. 실험은 0.03 의 contrast threshold 를 사용했을 때 생성되는 keypoint 수의 50% keypoint 를 발생했을 때를 기준으로, single threshold 를 사용했을 경우와 adaptive threshold 를 사용했을 때를 비교하였다. 이전에 실험결과에서 살펴본 바와 같이 다른 이미지들에 대해서도 adaptive threshold 를 사용한 결과가 10~15% 정도 keypoint 의 분포가 넓게 생성됨을 볼 수 있다. 또한 전체적으로 이때의 matching score 는 single threshold 나 adaptive threshold 를 사용한 경우 비슷한 결과를 보여준다.

표 3.5 입력 이미지에 따른 실험 결과

		Graffiti	Bikes	Boat	Leuven
Single threshold	Distribution	60%	70%	60%	57%
	Matching score	78%	90%	87%	90%
Adaptive threshold	Distribution	76%	83%	76%	67%
	Matching score	75%	90%	84%	89%

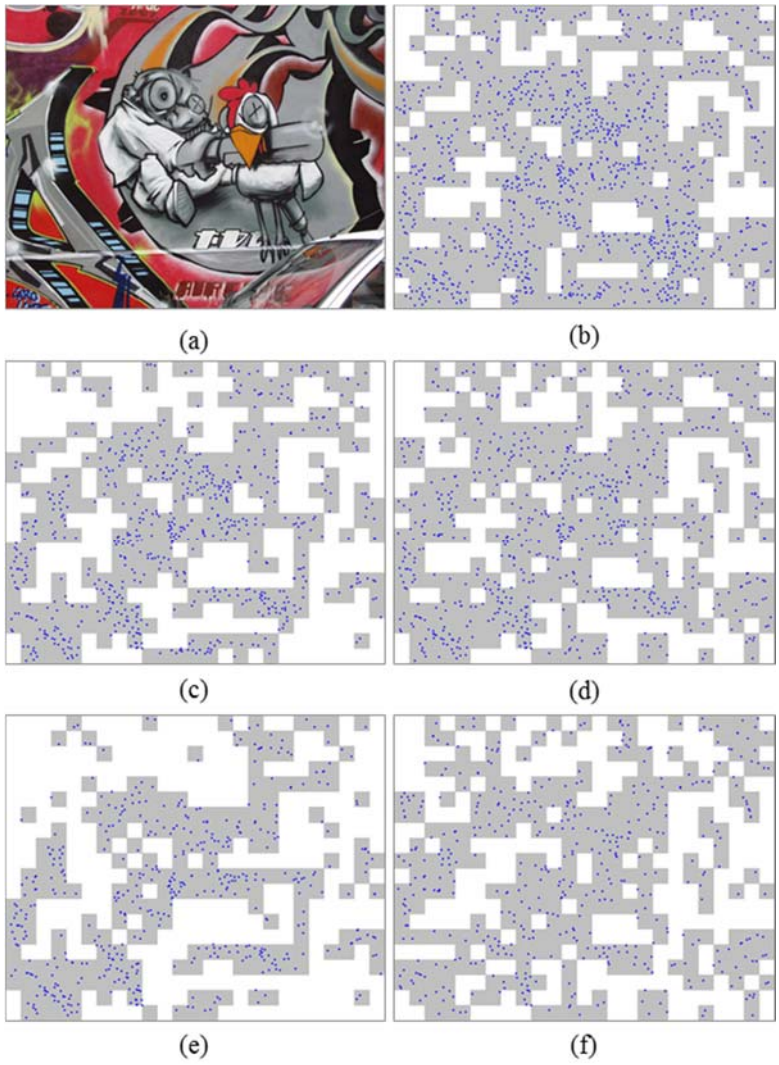
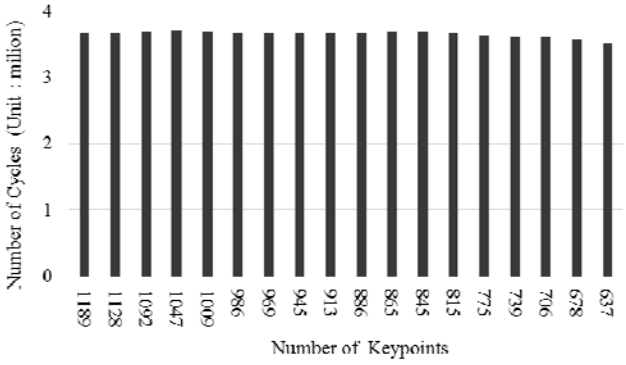


그림 3.19 Threshold 에 따른 keypoint 분포 (a) 실험 이미지 (b) 초기 keypoint 분포(n=1061) (c) Single threshold (n=756) (d) Adaptive threshold(n=756) (e) Single threshold (n=484) (f) Adaptive threshold(n=484)

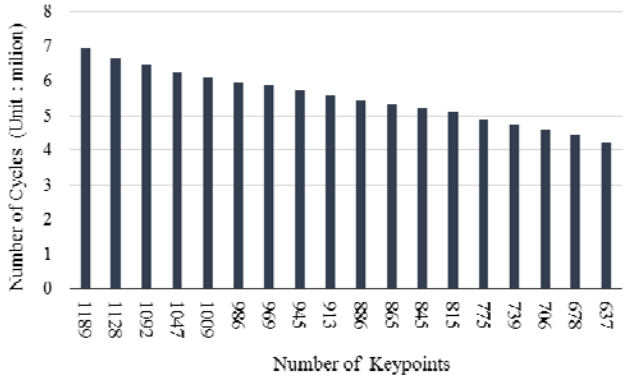
3.6.2 하드웨어 실험 결과

Adaptive keypoint 생성에 따른 keypoint detector 와 descriptor generation 의 수행 시간을 살펴보면 그림 3.20 (a) 와 같이 keypoint 수가 증가함에 따라서, keypoint detection 의 경우엔 일정한 수행 시간을 보이지만, descriptor generation 은 수행 시간이 줄어드는 결과를 볼 수 있다.

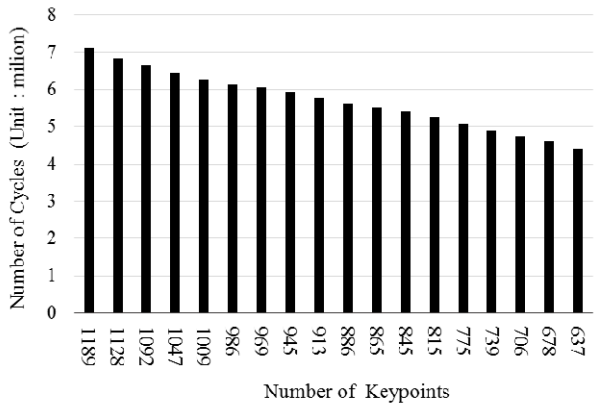
또한 전체적인 수행시간은 keypoint detector 와 descriptor generation 이 병렬로 진행되므로, keypoint detector 와 descriptor generation 수행시간의 합이 아닌 그림 3.20 (c) 와 같이 나타나고, 수행시간이 줄어드는 비율은 발생된 keypoint 수의 감소에 따라 descriptor generation 이 줄어드는 만큼 줄어들게 된다. 또한 전체적인 performance 관점에서, descriptor generation 부분이 keypoint detection 부분보다 더 많은 hardware 수행 과정을 필요로 하는 것을 볼 수 있다.



(a)



(b)



(c)

그림 3.20 Keypoint 수에 따른 하드웨어 수행시간 (a) Keypoint detection (b) descriptor generation (c) 전체 SIFT 수행시간

표 3.2 는 전체 하드웨어에 대한 합성 결과를 보여 준다. 합성 툴은 Synopsys 사의 Design compiler 를 사용하였다. Gate count 계산을 위해 target frequency 는 50Mhz 로 설정하고, silterra 0.13um library 를 사용하였다. 실험 결과는 SIFT 하드웨어에 adaptive keypoint 생성을 위해 추가된 'Block image characteristics generation' 부분이 전체 하드웨어의 gate count 에 비해 약 5 % 정도의 추가 logic 이 발생함을 보여 준다.

표 3.6 Adaptive keypoint generation 하드웨어 합성 결과

Module Name	Gate count	상대 비율
Space-space generation	478003	73.1%
Block image characteristics generation	**32526**	**5%**
Keypoint detection	49439	7.6%
Descriptor generation	94126	14.4%
Total	654094	100%

또한 제안된 하드웨어 에서는 초기 (96x32) 크기의 블록 이미지의 특성을 분석하기 위해서, 초기 latency 를 가지고 있다. 실험에서는 5932 cycle 이 초기의 latency 로 측정되었다. 초기의 latency 이후의 과정에서는 SIFT 의 Gaussian filter bank 블록과 블록내의 이미지 복잡도를 분석하기 위한 하드웨어가 병렬적으로 동작하기 때문에, 이로 인한 추가적인 수행시간의 증가는 발생하지 않는다. 이는 'Block image characteristics generation' 부분의 하드웨어 수행시간이 Gaussian filter bank 수행시간에 비해 빠르게 수행되므로

가능하다.

제안된 방법은 SIFT 하드웨어의 pipeline 구조가 깨지거나, 추가 하드웨어에서 외부와의 bus traffic 로 인한 delay 가 발생하지 않도록 설계되어 있다. 또한 블록단위로 수행하는 SIFT 의 내부메모리를 공유하는 구조로 설계가 되어 있다. SIFT 하드웨어는 블록 기반의 연산을 수행하고 있다. 따라서 제안한 방법은 adaptive 한 keypoint 조절을 위해서, 마찬가지로 블록 단위로 복잡도를 예측 가능하도록 되어 있다.

제4장 Region-constrained feature matching

서로 다른 이미지에 존재하는 물체를 인식하기 위한 방법으로 local feature 에 기반한 matching 방법이 사용되고 있다. 본 장에서는 여러 가지 feature matching 방법 중, hierarchical agglomerative clustering 에 기반한 matching 방법의 특징을 살펴보고, 이를 통해 inlier 와 outlier 를 효과적으로 구분하는 방법을 제안한다.

4.1 Hierarchical agglomerative clustering

Feature matching 을 수행하기 위해서, Euclidean distance 을 사용하여, descriptor 간에 유사성을 비교하는 방법은, local patch 의 유사성만을 비교하기 때문에, 부분적으로 유사한 다른 부분의 descriptor 로 correspondence 가 이루어 질 수 있다. 이는 많은 수의 잘못된 correspondence 가 포함될 수 있음을 의미하므로 descriptor 간의 유사성을 사용한 초기의 correspondence 로부터 correct correspondence(inlier) 와 incorrect correspondence(outlier) 를 구분하는 방법이 필요하다.

이전의 방법들 중 RANSAC 방법을 사용하여 affine transform model 을 추정하거나[9][11], geometric 한 정보를 사용하여, 이미지내의 keypoint 간 distance 나 angle 이 상대적으로 불변함을 이용하는 방법은[30][31][32] non-rigid 이미지에는 효과적이지 못하거나, 많은 수의 correspondence 가

존재하는 경우, 잘못된 keypoint 의 조합에 의해서도 distance 나 angle 간의 불변한 성질이 유지될 가능성이 높은 문제가 발생하게 된다. 따라서, 최근에 feature 간의 clustering 을 이용하여 신뢰성 높은 feature set 을 얻는 방법들이 있다[33]. 이와 같은 clustering 방법들은 이미지 내의 모든 feature correspondence 를 대상으로 모든 inter-cluster similarity 가 intra-cluster similarity 보다 클 때까지 반복적으로 수행한다.

그림 4.1 는 찾고자 하는 대상 이미지 그림 4.1 (a) 에 대해서 이미지가 변형이 되었을때 RANSAC 방법과 clustering 방법에 의해 matching 을 수행한 결과를 각각 그림 4.1(b), (c) 에 보여준다. 결과에서 볼 수 있듯이, 이미지의 변형이 심한 경우 RANSAC 방법으로는 object 를 정확히 찾기 어렵지만, clustering 의 경우엔 보다 정확한 결과를 얻을 수 있다.

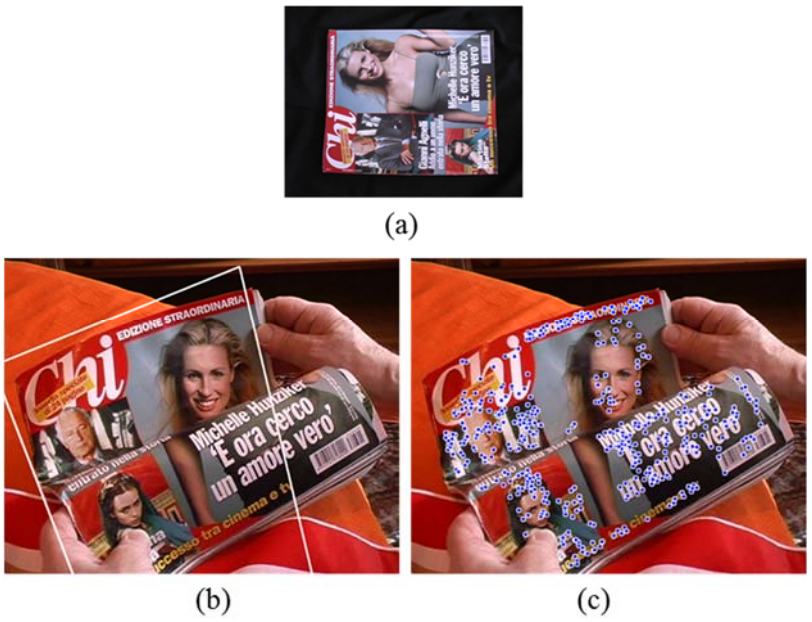


그림 4.1 Matching results (a) Model (b) RANSAC (c) Clustering

본 연구에서는 HAC(Hierarchical Agglomerative Clustering) 기반의 clustering 알고리즘을 사용한다[35][38]. 그림 4.2은 일반적인 clustering 방법을 보여준다. 첫 번째 단계는 (Correspondence extraction) 은 서로 다른 이미지에 존재하는 descriptor 간의 유사성으로 계산한 local feature 간의 초기 correspondence 를 계산하는 부분이다. 두 번째 단계는 (Cluster similarity) correspondence 들간의 similarity 를 계산하는 부분이다. 이와 같은 similarity 는 다음 단계(Clustering) 에서 clustering 을 위한 조건으로 사용된다. 두 번째 단계와 세 번째 단계는 모든 inter-cluster similarity 가 intra-cluster similarity 보다 클 때까지 반복적으로 수행된다.

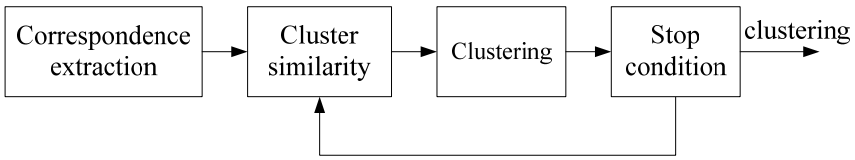


그림 4.2 A general flow of a clustering algorithm

HAC 는 clustering 방법 중 하나로 그림 4.2 과 같은 flow 를 가지고 있다. HAC 에 대한 전체 과정은 다음과 같다.

HAC Algorithm

Step 1: Determine all inter-correspondence similarities

Step 2: Select two closest correspondences or clusters and form a cluster

Step 3: Redefine similarities between the new cluster generated in Step 2 and the other correspondences or clusters

Step 4: Return to Step 2 until inter-cluster similarity is larger than intra-cluster similarity

Geometric similarity 를 계산하기 위해서, 먼저 두 개의 correspondence m_i , m_j 간의 distance 를 정의한다. 이때, p 와 q 을 각각 다른 이미지에 속해 있는 keypoint 라고 하고, 두 keypoint 는 h_i 에 의해서 correspondence 가 이루어져 있다고 하자. x_i 과 x_i' 가 각각 p 와 q 의 위치를 나타낸다고 하면, p 와 q 사이의 correspondence 는 $m_i = (x_i, x_i', h_i)$과 같이 표현할 수 있다. 이때 두 개의 correspondence $m_i = (x_i, x_i', h_i)$ 과 $m_j = (x_j, x_j', h_j)$ 사이의 distance 는 다음과 같이 정의한다[33].

$$
\begin{aligned}
d(m_i, m_j) &= \frac{1}{2}(d(m_j|m_i) + d(m_i|m_j)) \\
d(m_j|m_i) &= \frac{1}{2}(|x_j' - h_i x_j| + |x_j - h_i^{-1} x_j'|) \\
d(m_i|m_j) &= \frac{1}{2}(|x_i' - h_j x_i| + |x_i - h_j^{-1} x_i'|)
\end{aligned}
\tag{4.1}
$$

여기서 $|\cdot|$ 는 Euclidean distance 를 의미한다. 이와 같은 distance 에 대한 정의를 가지고, G 와 H 로 표현되는 두 개의 cluster 간의 similarity 는 식 (4.2) 와 같이 두 cluster 사이의 가장 가까운 correspondence 간의 거리로 정의된다.

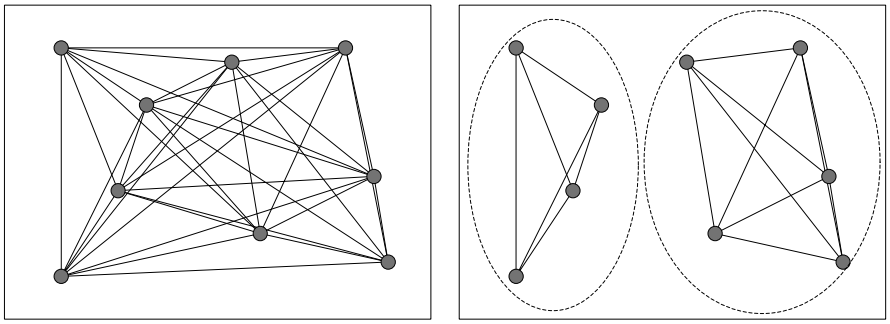
$$D(G,H) = \min_{\forall m_i \in G,\, \forall m_j \in H} d(m_i, m_j) \tag{4.2}$$

위의 정의를 통한 HAC 방법은 clustering 을 통해서, 효과적으로 inlier 와 outlier 간의 구분이 가능하지만, correspondence 의 수가 증가함에 따라 연산해야 할 데이터가 기하급수적으로 늘어나기 때문에, 급격하게 수행시간 증가가 발생하게 된다. 따라서 본 연구에서는 matching 의 정확도의 감소 없이 computational complexity 를 줄이기 위한 연구를 진행하였다.

4.2 Region-constrained clustering

본 연구에서는 효과적으로 clustering 을 수행하기 위해서, clustering 을 위한 candidate region 을 설정하여, candidate region 별로 clustering 을 수행하고, 그것들의 집합으로써 결과를 구성하는 방법을 사용하였다[34].

이번 절에서는 유사한 속성들로 이루어진 candidate region 을 생성하는 방법과, candidate region 내에 keypoint 를 가지고 clustering 을 수행하는 region-constrained clustering 방법에 대해서 설명한다.



(a) (b)

그림 4.3 Clustering 방법 비교(a) HAC (b) Region-constrained clustering

그림 4.3 는 HAC 방법과 region-constrained clustering 방법의 차이점을 보여준다. 그림 4.3 (b) 에 보이는 점선으로된 타원은 candidate region 을 표시한다. 두 가지의 방법은 동일한 keypoint 를 대상으로 하고 있다. 그러나 region-constrained clustering 방법은 candidate region 별로 candidate region 내의 keypoint 에 대해서만 clustering 을 수행하게 된다.

4.2.1 Geometric relationship in correspondence

P, Q 를 각각 두 개의 이미지로부터 생성된 keypoint 의 집합이라고 할 때, 두 개의 feature set P, Q 내의 feature vector p_u, q_v 는 식 (4.3) 과 같이 표현이 가능하다.

$$\begin{aligned} p_u &= [(x_u, y_u), \sigma_u, \theta_u, f_u] \\ q_v &= [(x_v, y_v), \sigma_v, \theta_v, f_v] \end{aligned} \tag{4.3}$$

여기서 (x_u, y_u) 는 keypoint 위치에서의 coordinate 를 의미한다. 또한 σ_u 와 θ_u 는 해당 위치에서 scale, orientation 정보를 나타낸다. f_u 는 descriptor 를 의미한다. 초기 correspondence 는 descriptor 들 간에 가장 가까운 Euclidean distance 와 두 번째로 가까운 Euclidean distance 를 비교하여, 계산된다. 이와 같은 초기 correspondence 를 가지고, correspondence 간의 homography matrix 를 계산할 수 있다. 각각의 feature vector 는 scale 과 orientation 에 대한 정보를 가지고 있다. 따라서 correspondence 에 대한 homography matrix (h)는 식 (4.4) 와 같이 scale 과 rotation, translation 정보를 가진 matrix 들의 곱으로 표현이 가능하다.

$$\begin{aligned} h &= [S(\Delta\sigma)][R(\Delta\theta)][T(\Delta x, \Delta y)] \\ &= \begin{pmatrix} \Delta\sigma & 0 & 0 \\ 0 & \Delta\sigma & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} cos\Delta\theta & -sin\Delta\theta & 0 \\ sin\Delta\theta & cos\Delta\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{4.4}$$

여기서 $\Delta\sigma = \sigma_v/\sigma_u$, $\Delta\theta = \theta_v - \theta_u$ 와 같이 정의되고, Δx, Δy 는 식 (4.5) 와 같이 표현이 가능하다.

$$\Delta x = x_v - (\Delta\sigma \cos\Delta\theta x_u - \Delta\sigma \sin\Delta\theta y_u)$$
$$\Delta y = y_v - (\Delta\sigma \sin\Delta\theta x_u + \Delta\sigma \cos\Delta\theta y_u) \quad (4.5)$$

또한 계산된 각각의 correspondence 의 homography 를 가지고, 이전 절에서의 식 (4.1) 과 식 (4.2) 를 통해서, correspondence 간의 similarity 를 계산할 수가 있다.

4.2.2 Constrained region

Region-constrained clustering 에서는 region 의 크기가 정확도와 computational complexity 에 영향을 미치게 된다. 좀 더 구체적으로는, region 내의 correspondence 수가 연산량과 밀접하게 연관되어 있다. 본 연구에서는 constrained region 을 설정하기 위해 segmentation 의 정보를 사용한다.

Segmentation 의 결과들은 때때로 over-segment 된 결과를 생성하게 된다. Over-segment region 에 의해서 생성된 영역들을 각각의 constrained region 으로 설정하면, 영역 내에 존재하는 correspondence 는 개수가 작을 뿐만 아니라, 영역 내에 존재하는 correspondence 들이 inlier 의 수보다 많은 outlier 로 구성되어 특성을 잘못 기술하는 경우도 발생하게 된다. 따라서, 본 연구에서는 segment region 들이 유사한 특성을 가지고 있을 때, over-segment 된 region 들을 병합하여, 가능한 많은 수의 유사한 특성을 가진 correspondence 를 candidate region 에 포함되도록 설정하는 방법을 제안한다. 본 연구에서는 segmentation 과정을 수행하기 위해서 watershed

transform 을 사용하였고 이에 기반하여, clustering 을 위한 후보 영역을 설정하였다. 특히 watershed segmentation 방법은 이미지를 edge 기반의 disjoint set 으로 구분할 수 있기 때문에, object 를 여러 개의 parts 로 나누는데, 사용할 수 있다. 일반적인 watershed 방법 및 블록 기반의 향상된 parallel watershed 방법에 대한 설명은 5 장에서 자세히 설명한다.

Segmentation region 들간의 유사성을 정의하기 위해서, 다음과 같은 region homography 를 정의하였다. Region 의 homography matrix 는 영역 내에 있는 각 correspondence homography 의 평균값으로 정의한다. 하나의 region A_i 에 m 개의 correspondence 가 존재한다고 할 때, A_i 의 area homography 는 식 (4.6) 와 같이 region 내에 평균적인 scale, rotation, translation 을 표현하는 각 matrix 의 곱으로 표현이 가능하다.

$$H_i = \mathrm{S}\left(\sum_{j=1}^{m} \Delta\sigma_j\right)\mathrm{R}\left(\sum_{j=1}^{m} \Delta\theta_j\right)\mathrm{T}\left(\sum_{j=1}^{m} \Delta x_j\,,\sum_{j=1}^{m} \Delta y_j\right) \qquad (4.6)$$

식 (4.6) 로 표현된 각 영역에 대한 region homography 를 사용하여, 인접한 두 개의 영역간의 homography 의 similarity 를 계산한다.

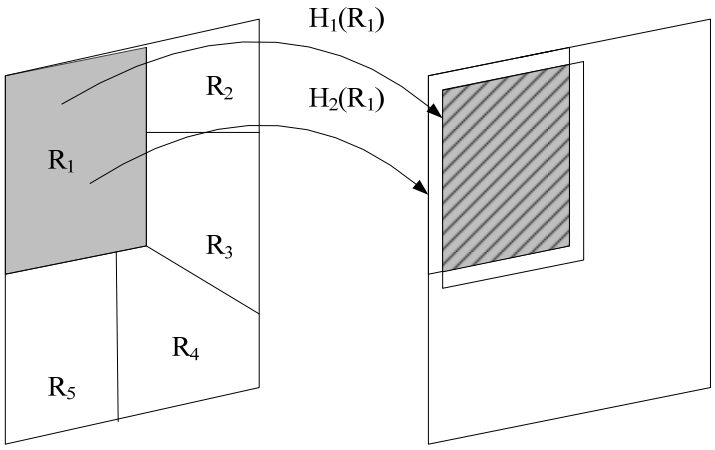


그림 4.4 Region homography projection

그림 4.4 에서 같이 인접한 두 개의 영역 R_1 과 R_2 에 대한 region homography similarity 를 생각해 보자. 식 (4.6) 에 의해서 계산된 영역 R_1 과 R_2 에 대한 homography 를 각각 H_1 와 H_2 라 할 때, 두 개의 homography 가 유사하다면, 하나의 영역을 기준으로 homography H_1 와 H_2 에 의해 projection 된 영역은 그림 4.4 과 같이 중첩될 것이다. 위와 같은 관찰을 통해서, 본 연구에서는 두 개의 homography 간 유사성을 측정하기 위한 방법으로 한 영역이 서로 다른 homography 에 의해서 projection 될 때의 overlap 된 영역의 존재 여부에 따라, homography 의 유사성 여부를 측정하였다. 이때, overlap 된 영역은 region 영역의 크기와 형태에 따라서, 영향을 받게 된다. 따라서, homography 만의 유사성을 측정하기 위한 수단으로 region 영역의 크기와 형태에 영향을 받지 않는 기준을 마련할 필요가 있다. 이를 위해서, 고정된 크기의 원 형태를(unit circle) similarity 측정을 위한 방법으로 사용하였다. 이와 유사한 접근 방법이 affine region

detector 에서 matching score 를 측정하는 수단으로도 사용되었다[20].

H_i 에 의해서 R_i 의 projected 된 영역은 식 (4.7) 과 같이 두 개의 matrix 의 곱으로 표현 가능하다. 여기서 matrix X_i^T 은 R_i 은 region R_i 내에 있는 모든 픽셀들을 나타낸다.

$$X_i^T = [x_i, y_i, 1] \;\; where\ for\ all\ (x_i, y_i) \in R_i \qquad (4.7)$$
$$X_{i'} = H_i X_i$$

위와 같은 관계를 사용해서, 각각 H_i, H_j 의 homography 를 가지는 두 개의 인접한 영역 R_i, R_j 의 region similarity 는 식 (4.8) 과 같이 표현 가능하다

$$Region\ similarity(R_i, R_j) = \begin{cases} 1, & H_i X_i \cap H_j X_i \neq \emptyset \\ 0, & H_i X_i \cap H_j X_i = \emptyset \end{cases} \qquad (4.8)$$

여기서 X_i 는 unit circle 내의 모든 픽셀들을 나타내는 행렬이다. 즉 $X_i^T = [x_i, y_i, 1]$ where for all $(x_i, y_i) \in UC$ 와 같은 관계를 가지고 있다.

위의 식에서 두 개의 projected region 이 overlap 되는 경우엔 “1” 의 region similarity 값을 가지고, overlap 되지 않는 경우엔 “0” 의 region similarity 값을 가지게 된다. Region similarity 가 “1” 인 영역들은 유사한 homography 를 가지는 영역들로 판단되어, 두 개의 영역들은 $R_1' = R_1 \cup R_2$ 의 새로운 영역으로 합쳐지며, 그 결과 clustering 과정에서 새로운 candidate region 이 생성된다.

영역별 clustering 의 정확도는 영역 내에 inlier 의 수가 outlier 의 수보다

클 때 높아진다. 제안된 방법의 candidate region 을 구성하는 방법은 over-segment region 에서 correspondence 의 수가 작은 경우에 region 병합에 의해서 정확도를 증가시켜 준다. 또한 region 별 clustering 을 수행할 때 두 개의 correspondence 들간의 similarity 를 계산하는 과정에서 correspondence 의 homography 를 사용하므로, 제안한 방법의 유사한 homography 들로 구성하는 candidate region 을 사용할 경우, 신뢰성 높은 결과를 얻을 수 있게 된다. Over-segment 된 이미지에서 모든 조합의 region 간 homography 를 조사하는 것은 많은 양의 연산을 필요로 한다. 따라서 이러한 computational complexity 를 줄이기 위해서, 인접한 영역간의 region 병합을 수행하게 된다. Segment region 에서의 인접한 영역들은 일반적으로 RAG(Region Adjacency Graph) 형태로 표현이 가능하다. 이와 같은 그래프를 사용하면, 인접한 영역간의 관계를 쉽게 알 수 있게 된다.

4.2.3 Complexity analysis

그림 4.5 는 입력 이미지를 segmentation 으로 나누고, clustering operation 을 위한 candidate region 을 구성하여, clustering 을 수행하는 과정을 보여준다. 초기의 segmentation 결과로부터 candidate region 을 형성하기 위해서, 초기 correspondence 로 계산된 region similarity 관계를 사용하며, 이를 통해 확장된 candidate region 이 생성된다. 또한, 이 candidate region 은 clustering operation 을 수행하기 위한 constraint 가 된다. 따라서, candidate region 별로 HAC 를 수행하게 된다. 최종 clustering 결과는 각각 candidate region 에서의 HAC 의 결과들의 합으로써 표현된다.

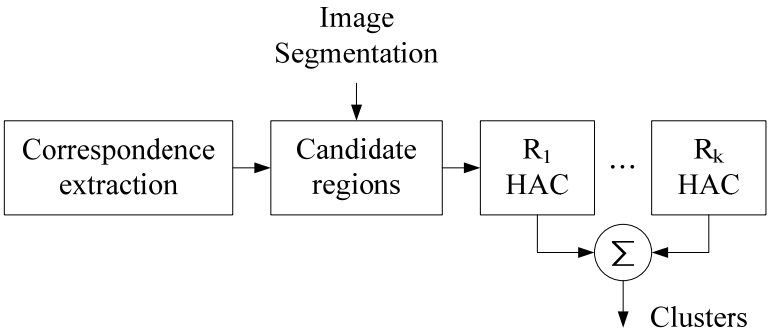


그림 4.5 The flow of the proposed clustering algorithm

N 개의 correspondence 에 대해서, 이전의 HAC 방법은 최대 N-1 개의 cluster 가 생성되며, 이 경우 HAC 알고리즘에서 설명했던 바와 같이 N-1 iteration 과정(step 2, 3, 4)이 필요하게 된다. 또한 cluster 간 similarity 를 계산하기 위해선, $O(N^2)$ 만큼의 연산이 필요로 한다. 따라서, 전체 complexity 는 $O(N^3)$ 이 된다[35]. 반면 k region 단위로 나누어진 이미지 내에서 각각 $n_1, n_2, \cdots, n_k$ 개의 correspondence 가 있다고 생각해 보면, 식 (4.9) 과 식 (4.10) 와 같은 수식이 만족된다. 따라서, 식 (4.10) 과 같은 관계에 의해서 제안한 방법은 이전 방법대비 수행 시간을 크게 줄일 수 있게 된다.

$$\bigcup_{i=1}^{k} n_i = n \,, where\ n_i \cap n_j = \emptyset \tag{4.9}$$

$$n_1^3 + n_2^3 \cdots n_k^3 \ll n^3 \tag{4.10}$$

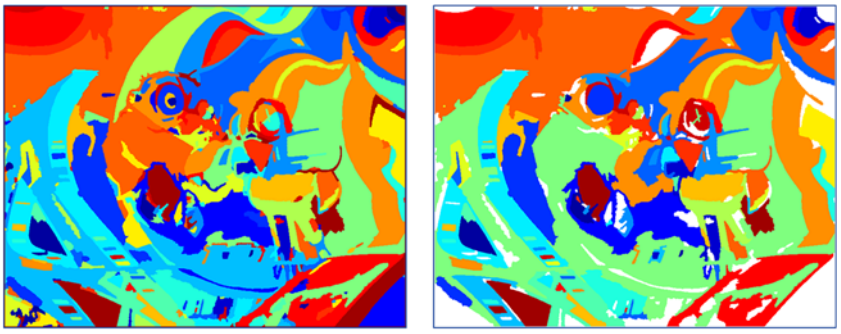
4.3 성능 평가

본 연구에서는 shared contents 를 가진 두 개의 dataset 에 대해서 실험을 진행하였다. 첫 번째 dataset 은 9 개의 model 과 23 개의 실험 이미지로 구성되어 있고[36], 또한 비교적 correpondence 의 개수가 적게 나오는 이미지들을 포함하고 있다. 두 번째 Oxford data set 은 viewpoint change, image blur, JPEG compression artifacts, illumination change 와 같은 다양한 degradation 에 의해 변형된 이미지들로 구성되어 있다. 또한 이미지의 크기가 크고, 복잡한 이미지들로 구성되어 많은 수의 correspondence 를 포함하고 있다. Oxford database 은 ground truth correspondence 를 제공하므로, 이를 통해 정량적인 matching socre 계산이 가능하다.

초기 correspondence 는 Lowe[9] 논문에서 사용한 NNDR 방법을 사용하여 생성하였다.

본 연구에서는 region 단위의 clustering 을 수행하고 있다. 그림 4.6 는 Oxford dataset 의 Graffiti 이미지에 대한 segmentation 결과와 이에 대한 candidate region 을 표시한다. 본 연구에서 사용한 watershed 에 기반한 segmentation 은 edge 를 유지하면서 object 의 part 단위별로 영역이 구분된다. 이러한 segmentation 결과에서 region 간 동일한 homography 를 유지하는 영역들을 하나의 영역으로 구성하였다. 그림 4.6 (a) 는 watershed 에 의해서 segmentation 된 영역들을 다른색들로 표현한 그림이고, 그림 4.6 (b) 는 각각의 candidate area 를 다른 색들로 표현한 그림을 보여준다. 초기의 segmentation 결과에 비해서, 인접 영역간 유사한 homography 를 가지는 영역들끼리 합쳐지는 것을 볼 수 있다. 흰색으로 표시된 영역은

correpondence 가 존재하지 않거나 하나의 correspondence 만 존재하는 영역을 보여준다. Clustering 은 correspondence 간 similarity 연산을 수행하기 위해서, 최소 2개 이상의 correspondence 를 필요로 하기 때문에, 이 영역은 clustering 연산에서 제외하게 된다.



(a) (b)

그림 4.6 Candidate areas for clustering over Graffiti image. (a) Segmentation areas (b) Candidate areas

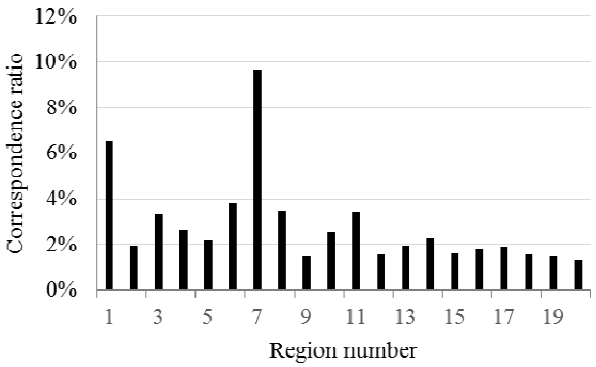


그림 4.7 전체 대비 candidate region 내의 correspondence 비율

그림 4.7 은 전체 이미지에서 발생하는 correspondence 의 개수 대비 candidate region 내에서 발생하는 correspondence 수의 비율을 보여준다. 그림은 이중 개수가 많은 상위 20 개의 영역들에서의 비율을 표시하고 있다. 모든 candidate region 들에서 correspondence 비율은 10% 보다 작다. 이와 같은 분포는 식 (4.10) 관계 따라 전체 수행시간을 크게 줄일 수 있게 된다.

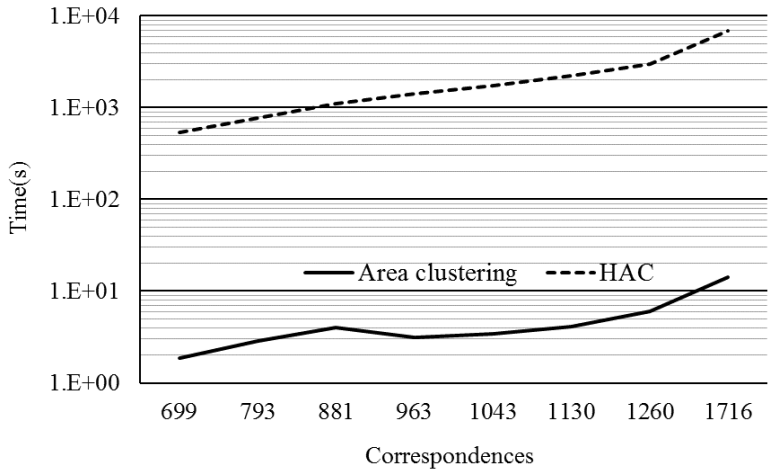


그림 4.8 Area clustering 과 HAC 의 수행시간 비교

그림 4.8 은 Oxford Graffiti dataset 을 사용해서, correspondence 수에 따른 수행시간을 보여준다. 가로축의 값은 NNDR threshold 를 달리했을 때 생성되는 correspondence 의 수를 표시하고, 세로축의 값은 log scale 로 표시된 수행시간을 보여준다. 실험은 2.67Ghz 로 동작하는 single-core Intel i5-750 processor 을 사용하였다. Correspondence 의 수가 증가함에 따라서, HAC 와 area-clustering 간의 수행시간 차이가 더 크게 증가하게 된다. 실

험에서 area-clustering 을 위한 수행시간은 segmentation 과정을 통한 candidate-region 생성시간을 포함하고 있다.

표 4.1 Recognition results over Oxford dataset

	Bikes	Graffiti	Boat	Leuven
Proposed cluster size	733	1086	917	1096
Proposed Matching score	72.7%	65%	83.2%	78.8%
HAC cluster size	886	1253	1062	1189
HAC Matching score	65.4%	58.9%	82.9%	74.9%

표 4.1 은 제안된 clustering 방법과 HAC 방법의 정확도를 보여준다. 정확도를 측정하기 위해서, feature matching 알고리즘에서 많이 사용되는 matching score 의 값을 사용하였다. 실험에서는 이미지 쌍당 수백 개의 initial feature correspondence 를 보이는 Oxford dataset 에 대해서 전체 이미지에 대한 clustering 방식을 적용했을 때의 결과와 area clustering 방식을 적용했을 때의 실험 결과를 보여준다. 실험에서 사용한 dataset 에서는 affine transform 값이 제공된다. 따라서, 정확한 matching score 값의 계산이 가능하다. 제안된 방법의 matching score 는 기존의 HAC 방법 대비 높다. 제안된 방법의 clustering size 는 각 candidate region 에서의 clustering 결과의 합으로 표현된다. 제안된 방법의 clustering size 는 HAC 의 방법의 결과

와 비교해서 줄어들지만, 높은 matching score 값을 가지고 있다. 이와 같은 결과는, 제안된 방법이 효과적으로 outlier 를 제거하는 것을 의미한다.

그림 4.9 그림 4.10 에 사용된 모델 이미지

그림 4.9 은 [36] 논문에서 사용한 dataset 에 대한 model 이미지를 보여준다. 이와 같은 model 을 사용했을 때, HAC 방법과 제안한 방법의 실험 결과는 그림 4.10 과 같다. 그림 4.10 (a), (c), (e), (g) 는 HAC 에 의한 실험 결과 이미지를 보여주며 (b), (d), (f), (h) 는 제안된 방법을 사용했을 때의 실험 결과를 보여준다. 결과에서 파란점은 각각의 방법을 사용해서, inlier 로 판단된 correspondence 에 해당하는 위치를 표시한다. Cluster 의 수는 제안된 방법으로 획득된 실험 결과에서 적게 나오지만, 제안된 방법의 경우 object 위에만 결과가 존재하는 것을 확인할 수 있다. 이는 area clustering 방법이 candidate area 의 영역별로 clustering 을 수행하기 때문에, candidate area 밖의 outlier 에 의해서 잘못 clustering 이 될 가능성이 낮다는 것을 의미한다.

그림 4.10 HAC versus area clustering (a), (c), (e) and (g) show the results by HAC. (b), (d), (f), (h) show the results by the proposed area clustering.

Recognition 의 정확도를 측정하기 위해서, recall 과 precision rate 를 사용하였다. Recall 과 precision 은 두 이미지 사이에서의 correct match 와 false match 의 수를 통해서 계산된다. Positive 나 negative 로 표현된 match 들은 4 가지의 조합, TP(True Positive), FP(False positive), TN(True Negative), FN(False Negative) 중 한 유형으로 분류된다. 이와 같은 유형들은 표 4.2 같은 matrix 형태로 표현 가능하다.

표 4.2 Confusion matrix

	Actual positive	Actual negative
Predicted positive	TP	FP
Predicted negative	FN	TN

이때, recall 과 precision 은 식 (4.11), 식 (4.12)과 같이 정의된다.

$$recall = \frac{TP}{TP + FN} \tag{4.11}$$

$$precision = \frac{TP}{(TP + FP)} \tag{4.12}$$

표 4.3 은 그림 4.10 의 precision 과 recall 값을 보여준다. 전체적으로 제안된 방법의 precision 값이 높다. 그러나 recall 의 값은 area clustering 방법이 candidate region 내의 correspondence 만을 가지고 수행하기 때문에, HAC 방법보다 낮은 값을 가진다.

표 4.3 그림 4.10 의 결과에 대한 recall 과 precision

	(a)(b)	(c)(d)	(e)(f)	(e)(h)
Proposed cluster size	51	79	340	100
Proposed recall	0.71	0.71	0.85	0.67
Proposed precision	0.96	0.95	0.99	0.95
HAC cluster size	78	124	408	120
HAC recall	0.81	0.92	0.97	0.70
HAC precision	0.72	0.78	0.95	0.83

이러한 방법은 전체 영역에 대해 clustering 을 수행한 기존 방법과는 달리 candidate region 단위로 연산이 이루어지기 때문에, clustering 을 위해 조사하는 correspondence 의 수가 region 내의 correspondence 로 제한되어, 적은량의 연산만으로도 계산이 가능하게 된다. 또한 영역 외부에 존재하는 outlier 의 영향을 받지 않기 때문에, 좀 더 정확한 결과 획득이 가능하게 된다.

제5장 Block based parallel watershed segmentation

많은 응용 분야에서 실시간 영상 분할을 필요로 하고 있으며, 여러 가지 영상 분할 알고리즘 가운데, watershed 알고리즘은 경계가 불분명한 이미지에서 효과적으로 이미지를 나눌 수 있는 있는 방법이다[40][1]. 이 방법은 이미지의 크기에 비례하여 수행 시간이 증가하기 때문에, 실시간 처리가 가능한 watershed에 대한 연구가 진행되어 왔다[41][42][43][44]. 이미지의 크기가 증가함에 따라 수행시간이 증가하는 문제점에 대한 개선 방법으로 이미지를 블록 단위로 분할하고, 병렬 연산을 수행하거나 일부 블록에 대해서만 연산을 수행하는 방법이 제안되었다. 블록 기반의 watershed는 각 블록 경계에서 영역이 분할되기 때문에 이를 통합하는 동작이 필요하다. 기존의 방법에서는 블록 기반 연산을 통한 영역 분할 결과가 프레임 기반 watershed보다 더 많은 영역으로 분할되는 문제가 있었다. 본 연구에서는 블록 경계에서 영역이 통합되거나 분리되는 경우를 관찰하여, 이 관찰을 근거로 경계 영역을 통합 방법을 제안하였다. 제안 방법은 watershed 및 label 매칭 동작이 블록별로 독립적으로 수행 가능하기 때문에 병렬 처리를 통하여 속도를 크게 향상시킬 수 있다.

5.1 Watershed transform

영상 분할은 이미지를 동질 영역들의 집합으로 나누는 기법으로 여러 영상 분석 방법들에 있어서 중요한 도구로 사용되며, 그 응용 분야 역시 매우 많다. 그 중에서 watershed 기반 영상 분할 방법은 edge 기반의 방법과 달리 영역의 경계가 희미한 경우에도 효과적으로 영역을 구분할 수 있기 때문에 가장 널리 사용되는 방법이다. 최근 여러 응용분야에서 영상 분할의 실시간 처리를 요구하는 경우가 늘어나고 있는데, 영상 분할 방법에서는 비교적 많은 연산이 수행되기 때문에 실시간 처리가 용이하지 않다. 따라서 실시간 영상 분할에 활용할 수 있는 watershed 알고리즘에 대한 연구가 이루어지고 있다.

본 연구에서는 Vincent-Soille watershed transform 을 사용하였다[39]. 이 방법은 이미지를 지형학적으로 고려하는데, 이미지의 gray level을 그 지점에서의 높이로 간주한다. Local 영역에서 가장 낮은 높이를 root라 정의하고 이 root에서 출발하여 영역을 병합해 나간다. 그림 5.1 은 이와 같이 영상을 지형학적으로 고려하는 개념을 설명하는 예를 보인다. Root와 인접한 픽셀들의 gradient가 단조 증가하는 경우 이 픽셀들은 root와 같은 영역에 있는 것으로 판단한다. 이러한 픽셀들을 기준으로 주변 픽셀들의 gradient를 조사하여 영역을 다시 확장하며, 이 영역은 동일한 root를 갖는다. 같은 root를 갖는 영역을 catchment basin (CB) 이라고 하고, 다른 root로부터 확장된 영역과 합쳐지지 않도록 하기 위해서 구분하는 경계선을 watershed line 이라고 한다. 이와 같이 영상 내에서 root들을 설정하고, 이 root들을 시작점으로 하여 전체 영상을 분할한다.

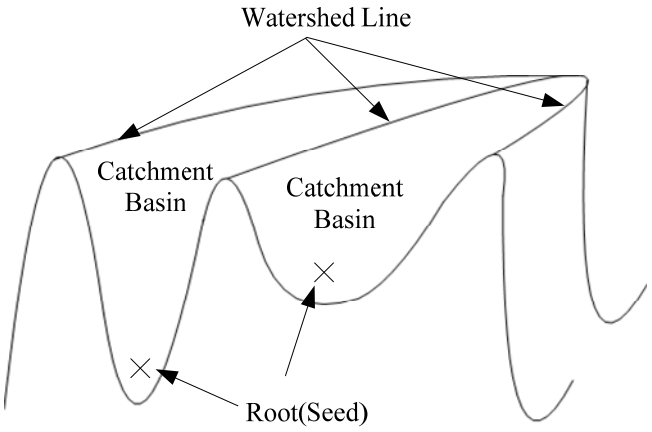


그림 5.1 Watershed 알고리즘 개념

Vincent-Soille watershed 알고리즘은 영상의 gradient 를 입력으로 받으며, 각각의 CB 이 동일한 label 을 가지는 disjoint set 결과를 출력한다. Watershed 알고리즘은 크게 두 단계의 과정으로 진행된다.

첫째, sorting 과정을 수행한다. Gradient 입력을 받으면 우선 gradient 를 기준으로 영상의 모든 픽셀들을 정렬한다. 이 과정은 가장 낮은 gradient 값(hmin) 으로부터 가장 큰 gradient 값(hmax) 값 순으로 정렬하고, 이후의 flooding 과정에서 특정 값의 gradient 에 해당하는 픽셀을 바로 접근하기 위해서 이러한 과정을 수행한다.

두 번째, flooding 과정을 수행한다. flooding 과정은 root 를 찾는 과정과, 이를 가지고 flooding 과정을 수행하여 CB 을 만드는 과정으로 나뉘어 진다. Root 는 해당 gradient 값이 주변의 다른 값보다 낮은 값을 가지거나 또는 주어진 높이 h 에서 주변의 모든 값들이 주변에 label 된 값을 가지지 않는 경우를 root 로 정의하게 된다. Flooding 과정에서는 현재 픽셀값이 이미

label 된 픽셀과 동일한 높이의 gradient 를 가지는 경우 현재 픽셀 주변의 이미 label 된 값을 부여하게 되고 이때, 같은 높이의 gradient 값을 가지는 동일한 label 들을 plateau 라고 하게 된다. 또한 현재 픽셀의 주변의 픽셀 값이 현재 픽셀보다 낮은 이미 label 된 값을 가지는 경우는 주변 픽셀의 label 값을 부여하게 된다. Flooding 과정에서 현재 픽셀의 주변의 픽셀의 label 값들이 서로 다른 경우에는 현재 픽셀을 watershed line 으로 정의한다.

5.1.1 Predictive watershed algorithm

이전 논문에서의 Predictive watershed algorithm[43] 은 Vincent-Soille 알고리즘의 속도 향상을 위해, 연속된 video frame 에서의 temporal correlation 을 사용하여 변화된 영역에서만 watershed 알고리즘을 수행한다. 변화된 영역의 계산은 블록 단위로 판단하기 때문에 변화된 영역이 블록의 집합된 형태로 나타나게 된다. 따라서 블록 경계에서 이미 계산된 영역(unchanged area) 과 새로 계산해야 하는 영역(update area)에서 경계 부분 처리 방법이 필요하게 된다.

Predictive watershed algorithm 은 Vincent-Soille 방법과 같이 update area 내에서 가장 낮은 gradient 값으로부터 주변 영역을 확장해 나가게 된다. 이때, 그림 5.2 (a) 에서 보는 것과 같이 update area 에 위치한 trilled ball 주변은 이미 계산된 Region 1 영역을 가지고 있다. 따라서 이 영역은 Region 1로 labelling 된다. 또한 update area 에 위치한 black ball 주변은 이미 계산된 Region 2 영역을 가지고 있다. 따라서 이 영역은 Region 2 영역으로 labelling 된다. 이와 같은 방법으로 그림 5.2 (b) 와 같이 3 개의 영

역으로 segment 된 region 을 생성하게 된다.

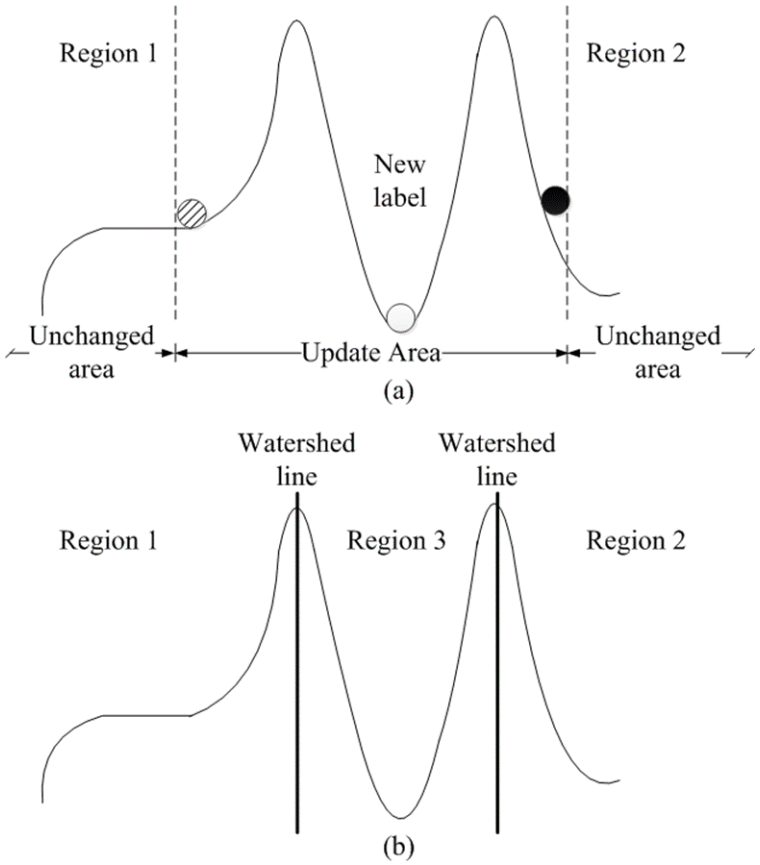


그림 5.2 Predictive watershed algorithm (a) Operation of the algorithm (b) Segmentation results

위와 같은 경우엔 변화된 부분의 블록에 대한 watershed 알고리즘이 성공적으로 수행되지만, 블록간 경계에는 다양한 경우의 수가 존재한다. 따라서 그림 5.3 (a)와 같은 gradient surface 를 갖는 영상에 대해서 predictive watershed algorithm 을 수행하면, Region 1 과 update area 의 경계에서는 주변 Region 1 의 label 이 전달되지만, update area 와 Region 3 경계에서는

update area 가 이미 가장 낮은 gradient 값으로부터 새로운 label 을 가지고 있기 때문에, 그림 5.3 (b) 와 같이 update area 와 Region 3 경계가 나누어지게 된다. 이와 같은 결과는 frame 기반의 watershed 에서는 2개의 영역이 생성되어야 하지만, 블록 기반의 연산의 결과로 over-segmentation 이 발생하는 경우가 된다.

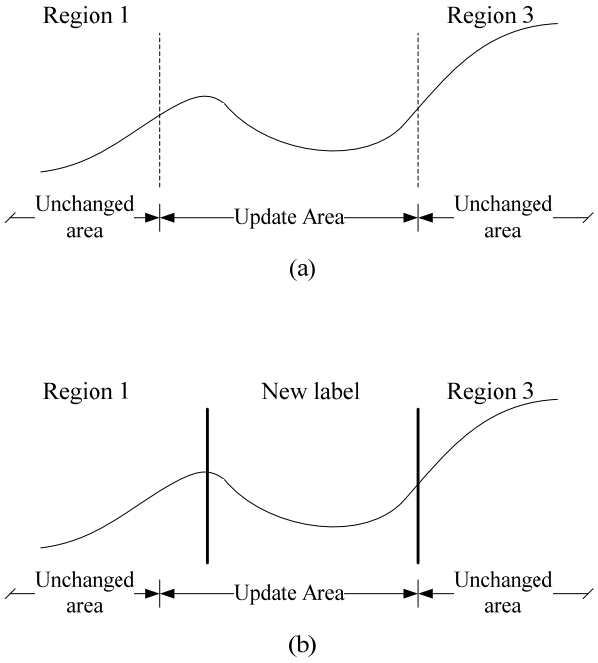


그림 5.3 Over-segmentation by predictive watershed algorithm (a) The gradient surface (b) Segmentation result

따라서 본 연구에서는 위와 같이 블록 경계에서 발생하는 over-segmentation 문제를 개선하기 위한 연구를 진행하였다.

5.2 Parallel watershed segmentation

본 절에서는 블록 기반 watershed 방법을 제안한다[41]. 그림 5.4 는 본 연구에서 사용한 segmentation 방법의 flow 를 보인다.

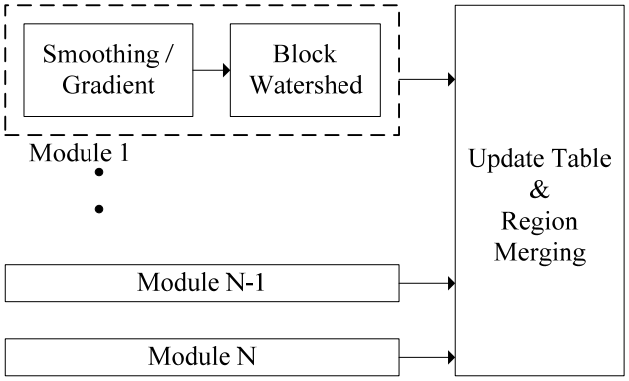


그림 5.4 Parallel watershed system

5.2.1 Preprocessing

영상이 입력되면 우선 이미지 smoothing 블록에서 입력 이미지의 노이즈를 제거한다. 이러한 동작은 노이즈에 의해 과도하게 영역이 분할되는 것을 방지한다. 입력 이미지에 대해 smoothing 을 수행한 후에는 이미지의 gradient 를 계산한다. 본 연구에서 사용된 watershed 알고리즘에서는 입력으로 영상의 gradient 를 사용한다. Gradient 는 일반적으로 object 의 경계에서 가장 큰 값을 가지며 이는 영상 분할에 유용하게 활용될 수 있다. 이와 같은 전처리를 거친 영상은 watershed transform block 으로 입력된다.

그림 5.4 에서 Image Smoothing 과정과 Gradient Calculation 과정은 픽

셀기반의 연산을 수행하므로 수행 시간이 많이 걸린다. 따라서 제안하는 블록 기반 watershed 의 효과를 극대화하기 위해서 이미지 smoothing 과 gradient calculation 의 전처리 과정도 블록 기반으로 수행하도록 하였다. 그림 5.5 는 블록 단위로 처리하는 순서를 보여 준다.

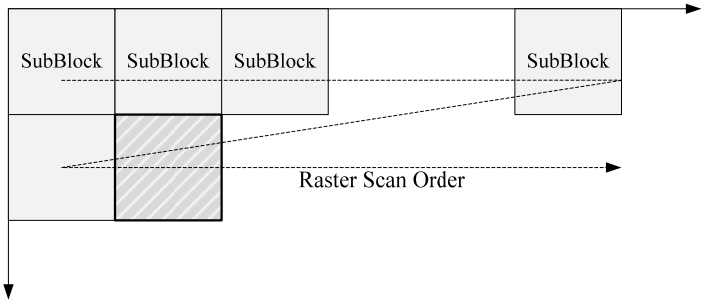


그림 5.5 Block processing order

이미지 smoothing 과정에서는 모든 element 가 '1' 인 3x3 윈도우를 가지는 mean filter 가 사용되었다. 블록 단위로 smoothing 을 수행하기 때문에 블록 경계 부분에서는 인접 블록의 픽셀 데이터가 필요하다. 그림 5.6은 블록 경계의 한 픽셀에 대한 smoothing 연산을 위해 필요한 인접 블록의 픽셀들을 보인다. 그림 5.6 에서 Block IV는 현재 처리하고 있는 블록을 의미하고, Block I, II and III 은 Block IV의 왼쪽 위, 위 및 왼쪽 블록을 나타낸다. P 로 표시된 점이 smoothing 을 하고자 하는 픽셀을 나타내고, 'X' 로 표시된 점은 P 의 연산을 위해 필요한 인접 블록의 픽셀을 의미한다. 따라서 smoothing 연산에서는 주변 블록의 1 픽셀폭에 해당하는 픽셀들의 값을 저장해야 한다.

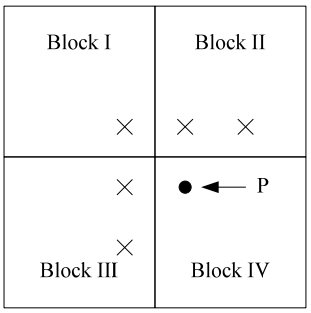


그림 5.6 Block connection

Gradient 계산에서는 morphological gradient 를 사용하였다. Morphological gradient 는 입력 이미지 f 에 대해서 morphological dilation 과 erosion 과정을 수행한다. $\oplus$ 과 $\ominus$ 을 각각 morphological dilation 과 erosion 연산으로 정의할 경우, morphological gradient 는 식 (5.1) 로 주어진다[46].

$$Gradient(f) = (f \oplus B) - (f \ominus B) \tag{5.1}$$

여기서 B 는 모든 element 가 1인 3x3 structuring element 를 나타낸다. Gradient 의 계산에서는 smoothing 과정과 마찬가지로 블록 경계에서 인접 블록의 1 픽셀폭에 해당하는 픽셀의 smoothing 이미지를 저장해야 한다. 이와 같은 블록 기반 전처리 과정을 거친 영상은 watershed transform block 의 입력으로 사용된다.

5.2.2 Block-based watershed segmentation

제안된 방법은 이미지를 블록으로 나누고 각각의 블록을 독립적으로 처리한다. 따라서, 하드웨어 구현에 적합하도록 되어 있고, 병렬 처리를 위한 확장이 가능하다. 이미지 단위로 segmentation 을 처리시에는 전체 이미지에 대한 smoothing 과정이 완료된 이후에 gradient 계산이 가능하고, 전체 이미지에 대한 gradient 가 완료된 이후에 watershed 알고리즘의 시작이 가능하다. 또한 이미지 크기에 해당하는 이전 결과들이 저장이 되어야 한다. 따라서 이에 해당하는 저장 공간이 필요하게 된다. 반면 블록 단위로 처리시에는 블록의 크기에 해당하는 저장 공간만을 가지고 처리하고 다음 블록을 처리하거나, 또는 동시에 동일한 모듈을 여러 개 두어 동시에 처리가 가능하기 때문에, distributed memory system 에도 적용이 가능하게 된다.

하지만 블록 단위로 watershed 를 수행했기 때문에 같은 영역이 여러 블록에 걸쳐 있을 때 블록 경계에서 영역이 분할되며, 그에 따라 이미지 단위의 watershed 보다 매우 많은 영역으로 나누어진다. 이미지 기반 watershed 와 동일한 수의 영역으로 분할하기 위해서는 블록의 경계에서 분할된 영역이 실제 object 의 경계인지 혹은 블록 단위 연산에 기인한 것인지를 조사해야 한다. 만약 블록 경계에서 인접한 두 영역이 실제로는 하나의 영역인 경우에는 두 영역의 label 을 같은 값이 되도록 해야 한다.

그림 5.7 는 하나의 object가 2개의 블록 (Block I, Block II) 에 나뉘어 위치하고, 그에 따라 블록의 경계에서 2개의 영역으로 분할된 예를 보인다.

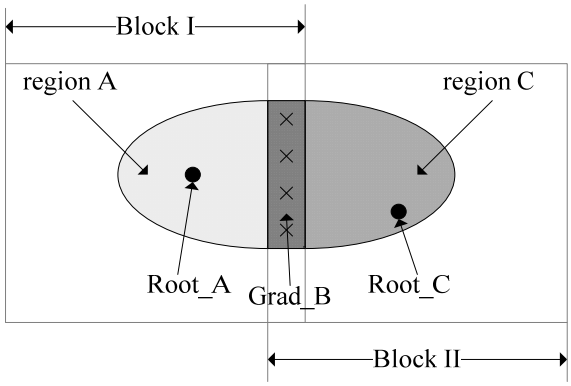


그림 5.7 Region across the block boundary

제안된 방법에서는 블록 단위의 연산을 위해 각 블록별로 Vincent-Soille watershed 알고리즘을 수행하고 그 결과를 각 블록마다 독립적으로 저장한다. Block I 에서 watershed 를 수행하면, region A 가 생성되는데 이 region A의 root의 gradient 값을 Root_A 라 한다. 그 다음 block II 에 대한 watershed 를 수행하는데, 이 때 이전 블록의 1 픽셀폭만큼의 픽셀들을 포함하여 watershed를 수행한다. 그 결과로 region C 가 분할되면 이 영역의 root 값을 Root_C 로 표시한다. 두 영역 경계에서는 1 픽셀폭만큼의 중첩된 영역이 발생하는데, 이 영역을 boundary area 라고 나타낸다. 그림 5.7 에서는 boundary 영역에서의 gradient 값을 Grad_B 로 표시하였다. 이 boundary area 에서는 두 개 sub-block 에서 수행한 watershed 결과를 동시에 가지게 된다.

이전의 연구에서는 블록 경계를 처리하기 위해 블록 경계에서의 추가 1 픽셀의 정보만을 사용할 경우, 이전 절에서 살펴본 바와 같이 over-segment 결과를 생성하게 된다.

본 연구에서는 블록의 경계에서 merge/split 판단을 하기 위해서, region 의 root 정보를 함께 사용하였다. Region 의 root 정보를 사용하는 것은 이를 저장하기 위한 추가적인 공간이 필요하고, merge/split 판단을 위한 추가적인 연산이 필요하지만, root 의 정보를 사용함으로써, 정확한 merge/split 판단이 가능하게 된다. Watershed 알고리즘에서 root 의 수는 label 의 수와 일치한다. 그림 5.8 은 Vincent-Soille watershed 알고리즘의 pseudo-code 를 설명하고 있다. Watershed 알고리즘은 입력으로 gradient 이미지를 사용하며, output 으로 픽셀 단위의 label 을 부여하는 동작을 수행한다. 이를 위해 영상내에 gradient 값을 sorting 하여, 낮은 gradient 값으로부터 순차적으로 주변 영역을 비교하는 과정을 수행하게 된다. 이때, local minimum 에는 새로운 label 을 부여하게 되는데, 이때의 local minimum 의 값이 root 값이 되게 된다. 따라서, 그림 5.8 에서와 같이 영역 label 에 대한 root 값은 watershed 연산 과정에서 저장이 가능하다.

Vincent-Soille watershed algorithm

```
1   Input    : Gradient Image
2   Output : labelled watershed image
3   /* SORT pixels of gradient values */
4   /* Start flooding */
5   for h=h_min to h_max do
6       output [P] = MASK;
7       if (output [ neighbors of P] !=0) then
8           enqueue (P);
9           distance (P) =1;
10      end
11      while (queue is not empty) then
12          /* Label P is a neighbor of other CB*/
13          /* Label P is plateau pixel */
14      end
15      if (output [P] is a MASK) then
16          output [P] = new label;
17          Root [new label] = h;
18      end
19  end
```

그림 5.8 Root information in Vincent-Soille watershed algorithm

Root 의 값은 그림 5.8 과 같이 표현된 Vincent-Soille 알고리즘에서 새로운 label 을 부여하는 과정에서 저장이 가능하다. 이와 같이 저장된 값을 가지고, 그림 5.7 과 같은 경계에서 root 의 gradient 값과 boundary 에서의 gradient 값을 가지고 발생하는 모든 경우에 대해 조사하게 된다. 그림 5.9 와 같은 경우의 수가 발생한다.

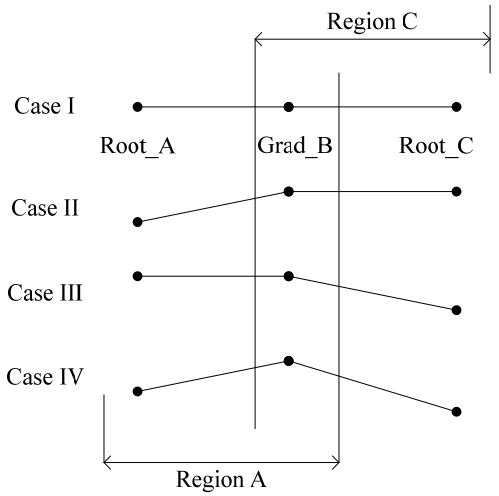


그림 5.9 Four possible relationship among Root_A, Grad_B and Root_C

Case I 은 Root_A, Grad_B, Root_C 의 값이 모두 같은 경우를 나타낸다. Case II 는 동일한 값을 가지는 Grad_B 와 Root_C 보다 Root_A 값이 작은 경우를 나타낸다. Case III 는 Root_A 와 Grad_B 가 동일하고 이 값이 Root_C 보다 큰 경우를 나타낸다. Case IV 는 Grad_B 의 값이 Root_A 와 Root_C 보다 큰 경우를 나타낸다. 이때, Root_A 와 Root_C 간의 크기는 관계가 없다. 왜냐하면 이것은 Root_A 와 Root_B 는 watershed 알고리즘의 특성상 식(5.2) 와 같이 항상 Grad_B 보다 작거나 같기 때문이다.

$$\begin{aligned} Root\_A &\leq Grad\_B \\ Root\_C &\leq Grad\_B \end{aligned} \tag{5.2}$$

그림 5.9 에서 보이는 4 가지 경우는 경계면에서 발생하는 모든 경우를 나타낸다. 따라서, 이와 같은 경우들에 대해 merge/split 을 정확히 수행하면

over-segment 문제 해결이 가능하다. 표 5.1 은 위와 같은 경우에 대해서, 판단하는 방법을 정리해서 보여준다. Case I, II, III 는 다른 블록내에 있는 두 개의 영역이 합쳐져야 하는 경우를 나타낸다. Case IV 는 두 개의 영역이 분리되어야 하는 경우를 보여준다. 이때, Case II, III 는 각각 두 개의 영역을 하나로 합치면서, root 의 값은 해당 영역의 가장 작은 값 Root_A, Root_C 로 update 가 필요하게 된다. 그림 5.10 은 이와 같은 블록간 region merge/split 을 수행하기 위한 pseudo-code 를 나타낸다. 이 과정은 입력으로 블록간 segmentation 된 결과를 사용하고, output 으로 병합되어야 하는 label 간 lookup table 을 생성하게 된다.

표 5.1 Decision for region merge/split

Case	Condition	Decision	Updated root
I	Grad_B = Root_C Grad_B = Root_A	Merge	Root_A or Root_C
II	Grad_B = Root_C Grad_B > Root_A	Merge	Root_A
III	Grad_B > Root_C Grad_B = Root_A	Merge	Root_C
IV	Grad_B > Root_C Grad_B > Root_A	Split	-

Algorithm 2 : Region Merging Criteria

Input : For Reference / Current Block
Root of region / Region Label
Boundary Gradient
Output : Region Lookup Table

```
1   If (root (C) == Gradient(B)) then
2   |  If (Gradient(B) == root (A)) then
3   |     Merge (A,C) and update table;
4   |  Else if (Boundary Gradient > root(A)) then
5   |     Merge(A,C);
6   |     root (C) = root (A);
7   |  end
8   end
9   If (root (C) < Boundary Gradient) then
10  |  If (root(A) == Boundary Gradient) then
11  |     Merge (A,B) and update table;
12  |     root(A) = root(C);
13  |  Else
14  |     Boundary pixel is WSHEDline;
15  |  end
16  end
```

그림 5.10 Region merging algorithm flow

5.2.3 Block-based skip of flooding operations

본 연구에서 제안한 방법은 watershed 연산을 독립적인 블록 단위로 처리한다. 이러한 블록 단위로 영상을 분할하는 경우 일부 블록 내에서는 영상이 분할되지 않고 블록 전체가 하나의 영역이 될 수 있다. 이러한 블록에서는 watershed 연산 과정 중에서 flooding 연산을 skip 하여 연산량을 줄일

수 있다. 이는 연산량을 효과적으로 감소시킬 수 있는 방법이 된다. 이와 같이 블록이 하나의 segment 만으로 구성되는 경우는 블록내에 root 의 값이 1개이거나 블록내의 모든 gradient 값이 동일한 경우이다. 이와 같은 판단은 블록내의 gradient 값을 sorting 하는 단계에서 가능하다. 그림 5.11 은 flooding 연산 skip 을 위한 flowchart 를 보여준다.

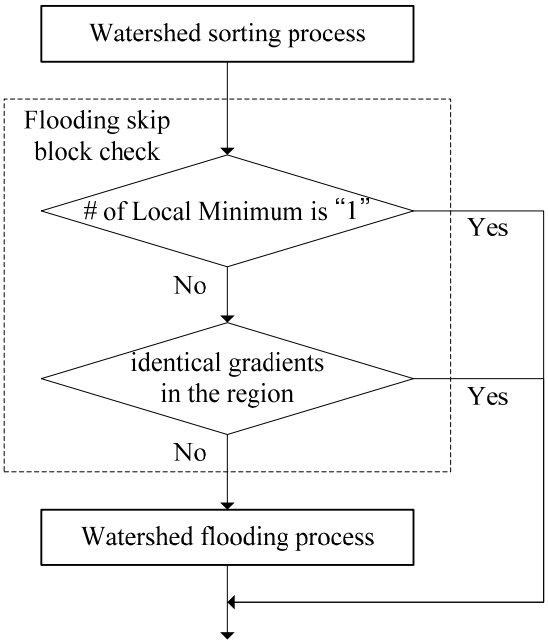


그림 5.11 Decision for skipping the flooding operation in a block

5.2.4 Update table and region merging

이 단계에서는 region lookup table 을 이용하여 label 을 정리하는 과정을 수행한다. Region lookup table 은 2 개의 column으로 구성되며 서로 merge

되어야 하는 영역들 간의 관계를 표현하고 있다. 두 개의 영역간 관계를 표현하는 table 에서 낮은 값의 label 을 첫 번째 column 에 위치시키고, 여러 단계의 연결 관계에서 경로압축(path compression) 방법으로 최종적으로 사용할 label 을 가르치도록 table 을 update 하게 된다. 그림 5.12 는 이와 같은 과정의 예를 보여준다. 그림 5.12 (a) 는 초기 영역간 관계를 저장하고 있다. 이와 같은 영역 관계에서 label 2 는 label 1 와 merge 가 되어야 하는 영역을 표시하고, label 3 는 label 2 와 merge 되어야 하는 영역을 표시한다. 따라서 label 1,2,3 은 하나의 label 을 가져야 한다. 최종적으로 정리된 lookup table 은 그림 5.12 (b) 와 같게 된다.

No	1st column	2nd column
①	1	2
②	2	3
③	1	4
④	6	7
⑤	1	6

(a)

No	1st column	2nd column
①	1	2
②	1	3
③	1	4
④	1	7
⑤	1	6

(b)

그림 5.12 Example of arranging region label (a) Initial lookup table (b) Arranged lookup table

정리된 region lookup table 을 통해 서로 병합되어야 하는 영역의 label update 를 수행하게 된다.

5.3 성능 평가

이전의 Vincent-Soille watershed 알고리즘은 전체 영상의 global 정보를 사용하기 때문에, 블록 단위로 나누어서 병렬처리하는데, 어려움이 있다. 본 연구에서 제안한 블록 기반의 접근 방법은 하드웨어 구현에 적합하도록 각각의 블록이 독립적으로 watershed 를 수행하도록 되어 있다. 또한 블록 단위로 연산이 이루어지기 때문에, 하나의 segment 로만 구성되는 블록이 발생하게 되고, 이를 사전에 예측하여, watershed 에서의 flooding 과정을 수행하지 않음으로써, 연산량 감소가 가능하다.

제안한 방법의 watershed 결과는 Vincent-Soille 알고리즘을 사용한 watershed 의 결과와 동일한 region 개수를 가진다. 그림 5.14 은 4 개의 영상에 대해서 Vincent-Soille watershed segmentation 방법과, 제안된 방법을 적용하였을 때의 block based segmentation 결과를 보여준다. 결과에서는 영역의 분할된 위치가 다르게 분할된 부분이 있는데, 그림 5.13 은 이 차이를 설명하는 예를 보인다[45]. 그림 5.13 에서 sub-block I 에 존재하는 root 의 위치를 흰색공으로 나타내고, sub-block II 에 존재하는 root의 위치를 검은색공으로 표시한다. 두 블록 경계는 동일한 gradient 를 갖는 plateau 위에 있다. Vincent-Soille 알고리즘을 수행할 때에 데이터의 scan 순서, 즉 sorting 후에 FIFO 에 저장된 데이터의 순서는 좌측에서 우측으로 처리되는 것으로 가정한다. Vincent-Soille 알고리즘의 flooding 과정을 수행하면 흰공으로부터 시작된 영역이 ① 의 위치에 도달하게 되고, 검은색공으로부터 시작된 영역은 ② 의 위치까지 flooding 과정을 수행하게 된다. Plateau area 에서는 동일한 gradient 값을 가지므로 FIFO 에 저장된 순서인

① 의 경계에 있는 픽셀들부터 주변영역과 비교하며 영역을 확장해 나가게 된다. 따라서 흰색공으로부터 출발한 영역으로 통합되게 되고, ② 의 경계까지 영역을 확장하게 된다. 따라서 최종적으로 ② 의 경계에서 Vincent-Soille 알고리즘의 watershed line 이 발생하게 된다. 제안한 방법에서는 case IV에 해당하기 때문에 영역이 분할되는데, 블록의 경계에서 watershed line 이 발생한다. 만약 ③ 의 위치에 블록의 경계가 존재하게 되면, ③ 의 위치에서 watershed line 이 발생하게 된다. 따라서 Vincent-Soille 알고리즘 기반 watershed 와 제안한 블록 기반 watershed 의 결과에 차이가 발생한다. 그러나 동일한 gradient 값을 갖는 plateau 은 영상내에서 blurring 이 심하거나, 영역분할을 위한 경계가 명확하지 않은 부분이다. 따라서 이러한 차이는 큰 문제를 발생시키지 않는다.

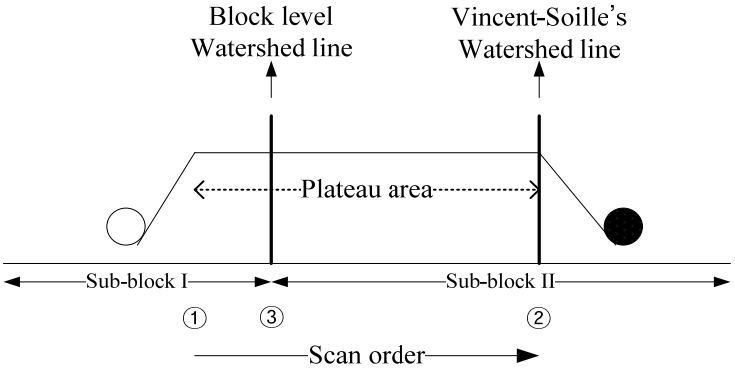


그림 5.13 Vincent-Soille 방법과 제안된 방법의 경계부분 차이

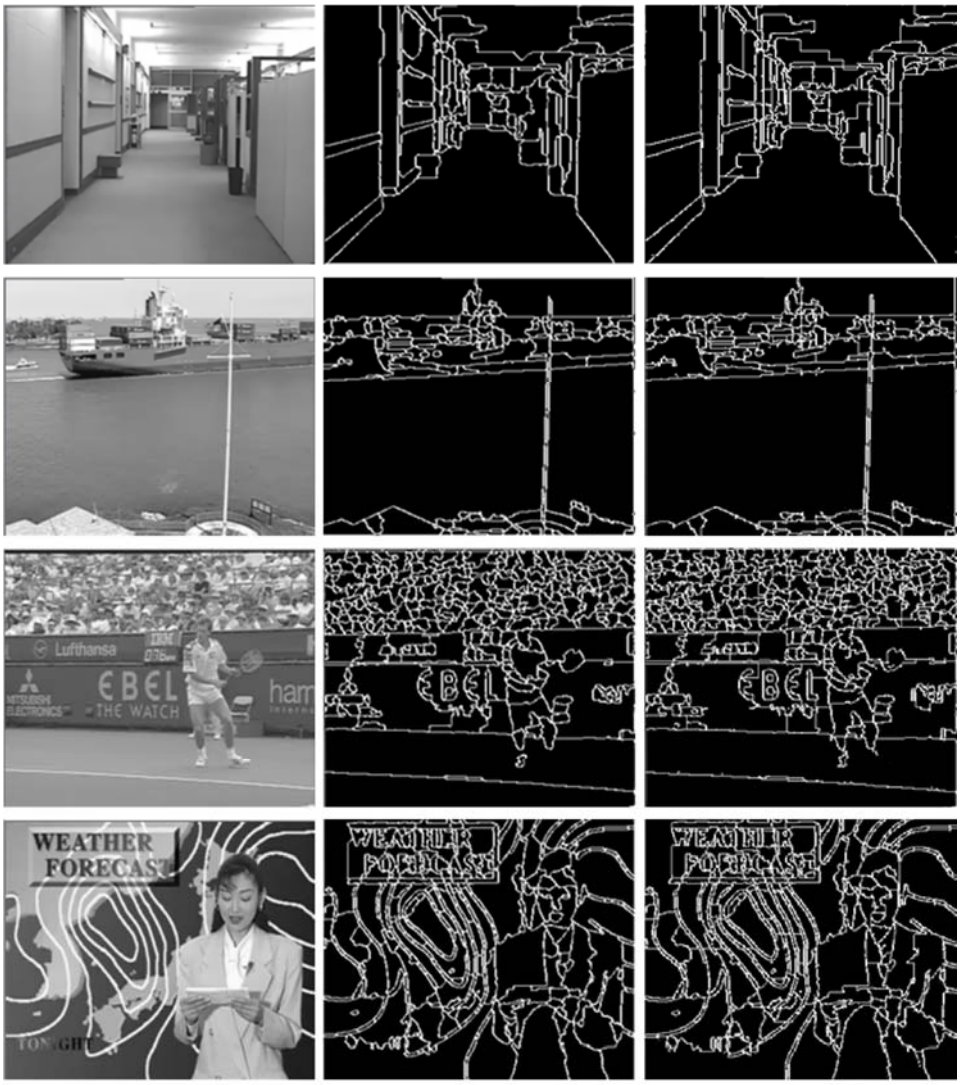


그림 5.14 Segmentation result (a) original image (b) Segmentation by Vincent-Soille algorithm (c) Segmentation by proposed algorithm

그림 5.15 는 제안한 방법을 프로세서의 수를 달리해서, 수행하였을 때의 결과를 보여준다. 각각의 processor 에서 계산된 결과는 boundary 를 계산하기 위한 연산과정을 거치기 때문에 최종적인 수행시간은 두 개의 processor 를 사용했을 경우 하나의 processor 를 사용하는 경우 대비 1.9 배의 속도 향상이 발생하고, 네 개의 processor 를 사용했을 경우 3배 정도의 속도가 향상된다.

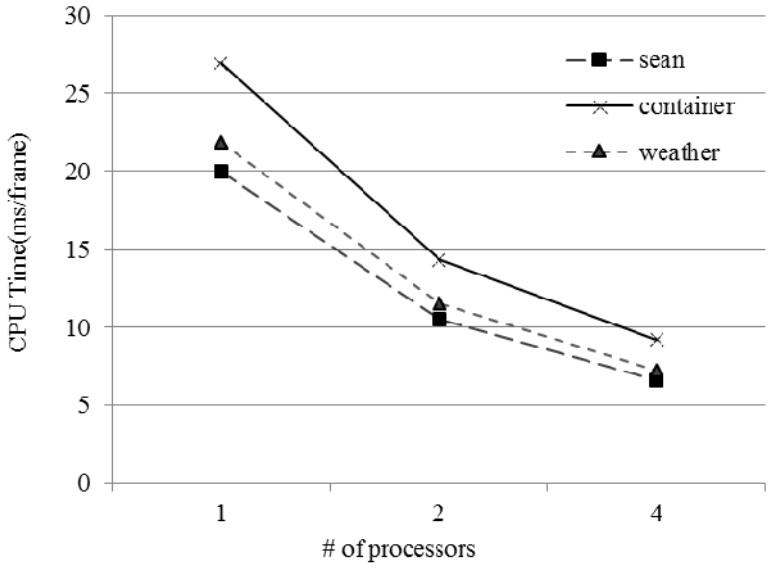


그림 5.15 Computation time versus the number of processors

표 5.2 는 16x16 의 블록 단위로 제안된 방법을 수행했을 때 flooding 알고리즘을 skip 하는 블록의 비율을 나타낸다. 이는 단일 영역으로 segment 되는 영역이 많이 분포되어 있는 영상에서 높게 나타난다.

표 5.2 Flooding operation skip 발생 비율

	Container	Foreman	Hall monitor	Sean	Weather
Flooding skip block	21%	6%	10%	7%	10%

표 5.3 은 전체 watershed 알고리즘의 전체 수행 시간 대비 flooding operation 의 수행 시간의 비율을 보여준다. Flooding 연산은 하나의 픽셀에 대해서, 주변의 여러 픽셀들을 조사하는 과정을 필요로 하기 때문에, sorting 과정에 비해서 많은 연산을 필요로 하는 부분이다. 실험적으로 5 개의 sequence 에 대해서 약 90% 정도의 수행 시간을 flooding 과정에서 필요로 한다. 따라서, 제안된 flooding skip 방법은 전체 연산량을 줄일 수 있는 효과적인 방법이 된다.

표 5.3 Execution time of flooding skip effect

	Container	Foreman	Hall monitor	Sean	Weather
Flooding operation ratio	89.5%	90.5%	89.9%	90.5%	92%

이전의 predictive watershed 알고리즘은 전체 영상을 블록으로 나누지 않고, 영상내에서 변화된 부분을 찾고 변화된 부분을 하나의 단위로 watershed 를 수행하는 방법이다. 따라서, predictive watershed 와의 비교를 수행하기 위해서, 연속된 프레임에서 변화된 부분에 대해 predictive

watershed 와 제안된 방법을 사용한 결과를 비교하였다. Predictive watershed 에서는 변화된 영역의 연결된 블록들을 하나의 단위로 연산하는 방법을 사용하였다. 그림 5.16 (a) 은 Hall monitor 영상의 20번째 프레임을 보여주고, (b) 는 해당 프레임에 대한 Vincent-Soille 알고리즘의 결과이다. 그림 5.16 (c) 는 이전 프레임에서 변화된 부분을 회색 부분으로 나타낸 것으로 회색 부분은 새롭게 watershed 를 수행해야 할 영역을 의미한다. 이와 같은 영역에 대해서 그림 5.16 (d) 는 predictive watershed 의 결과이고, (e) 는 제안한 방법의 결과를 보여준다. 제안한 방법에서 보다 정확한 segmentation 결과를 보임을 확인할 수 있다.

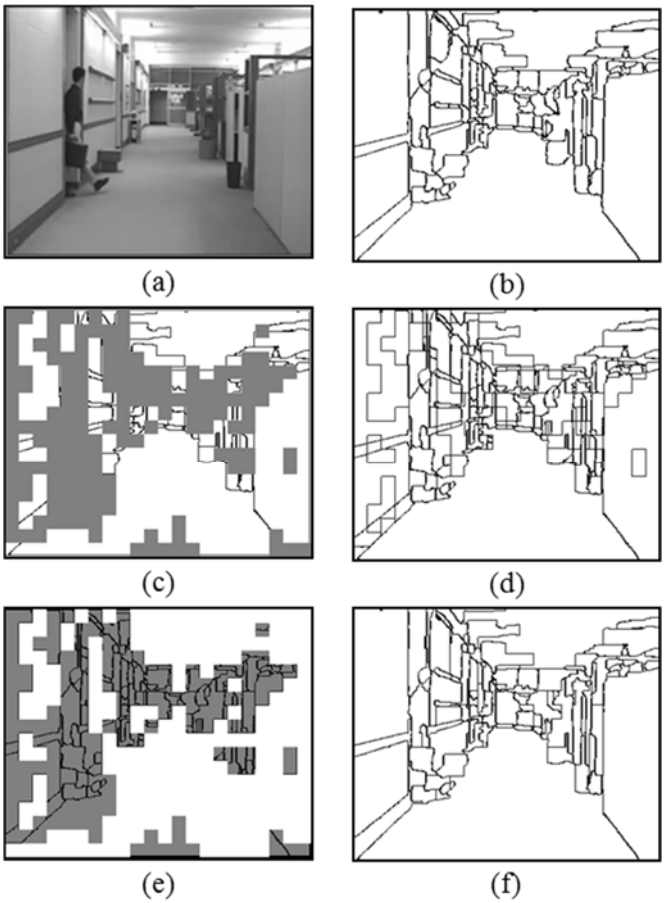


그림 5.16 Watershed segmentation results; (a) Hall Monitor sequence, (b) segmentation by Vincent-Soille algorithm, (c) updating area, (d) segmentation by predictive algorithm, (e) segmentation inside the updating region, (f) segmentation by the proposed algorithm.

제6장 결론

본 연구에서는 영역 분할 기반의 고속의 feature matching 방법을 통해 효과적으로 물체를 인식하기 위한 연구를 진행하였다. Feature matching 을 통한 물체 인식 방법은 여러 단계의 과정을 수행한다. 본 연구에서는 각각의 단계별로 성능을 향상시키기 위한 방법을 분석하고, 개선 방법들을 제안하였다.

첫 번째로, SIFT 하드웨어 에서의 속도 향상 방법으로 adaptive SIFT keypoint 생성 방법을 제안하였다. 기존에 keypoint 를 조절하는 방법들은 발생된 keypoint 를 post-processing 과정을 통해 sampling 하는 방법을 사용하였다. 그러나 keypoint detector 와 descriptor generator 가 병렬화된 hardware 에서는 병렬 구조를 유지할 수 없기 때문에, 이와 같은 방법을 적용하기가 어렵다. 또한 SIFT 의 contrast threshold 를 조절하여, keypoint 의 수를 줄이는 것은 contrast 가 높은 영역에만 keypoint 가 존재하여, 일부 영역에서는 특징을 기술할 특징점 자체가 없어지는 문제가 있다.

본 연구에서는 제안하는 방법은 블록 단위로 영상의 복잡도를 예측하여, 블록의 특성에 따라서 keypoint 가 상대적으로 많이 나오는 영역의 keypoint 는 조절하고, keypoint 가 적게 나오는 영역에서는 keypoint 가 유지되도록 하는 방법을 사용하였다. 이에 따라서, 전체적인 keypoint 의 분포를 고르게 유지시킬 수 있다. 이와 같은 keypoint 발생갯수는 application 에 따라 조절이 가능하다. 블록 내의 영상의 복잡도 예측은 FAST corner

detector 를 사용하였다. 이를 위해 영상의 복잡도를 예측하기 위한 hardware 를 설계하고, SIFT hardware 와 연동하여 동작하도록 하였다.

이 하드웨어는 SIFT 의 hardware 대비 추가적으로 5% 정도의 logic 을 필요로 하며, 전체 SIFT 의 Gaussian filter bank 연산 과정에 비해 빠른 처리가 가능하기 때문에, 초기의 latency 이후에는 SIFT 의 동작과 병렬 처리가 가능하다. 또한 이와 같은 방법에 의한 keypoint generation 방법은 전체 영상을 단일 contrast threshold 로 keypoint 를 조절하는 방법에 비해, keypoint 가 고르게 분포되어 있으며, matching score 측면에서도 이 방법을 적용하기 전과 동등 수준의 matching score 를 유지하고 있음을 확인할 수 있었다.

두 번째로, region-constrained feature matching 방법을 제안하였다. Local patch descriptor 의 유사성만을 이용하여, 대응 관계를 계산하면, 부분적으로 유사한 다른 부분의 descriptor 로 correspondence가 이루어질 수 있다. 따라서 이러한 초기의 correspondence 로부터 inlier 와 outlier 를 구분하기 위한 방법으로 HAC(Hierarchical agglomerative clustering)기반의 matching 을 수행하였다. 그러나 이 방법은 전체 영상에 대한 모든 correspondence 에 대해 clustering 을 수행하므로, 많은 수의 연산량을 요구한다. 본 연구에서는 이를 개선하기 위해, clustering 을 수행하기 위한 region 영역을 설정하고, 설정된 region 내에서 clustering 을 수행하였다. 초기의 segment 된 영역에서 region 간 homography 가 유사한 영역들을 합치는 방법으로 후보 영역을 구성하였고, 이를 통해 clustering 의 정확도를 향상할 수 있었다. 따라서 제안한 방법은 기존 HAC 방법에 비해 높은 matching score 를 가지

며, 1000 개의 correspondence 를 가지는 이미지에서 500 배까지 수행 시간을 단축을 확인할 수 있었다.

세 번째로, 블록 기반의 parallel watershed segmentation 방법을 제안하였다. Watershed 알고리즘은 전체 영상의 값에 대하여 순차적으로 연산을 수행하기 때문에, 병렬화가 어렵다. 또한 영상의 크기가 커짐에 따라 watershed 의 수행 속도가 급격히 저하되므로 실시간 처리가 매우 어렵게 된다. 따라서, 영상을 블록 단위로 나누어 처리하는 방법들이 연구되었는데, 이전 방법에서는 Vincent-Soille 알고리즘의 결과와는 달리 블록간 경계 부분에서 영상이 과분할되는 문제점을 가지고 있다.

본 연구에서는 watershed 수행 단위를 일정 크기의 블록으로 나누고, 블록간 독립적인 연산이 가능한 방법을 제안하였다. 블록간에 발생하는 dependency 는 블록 경계면에서의 gradient 값과 블록간 인접하고 있는 segment 의 root 값의 관계를 통해 예측이 가능하다. 블록간에 발생할 수 있는 모든 경우의 수를 처리함으로써 Vincent-Soille watershed 의 결과와 동일한 개수의 segment 가 생성 가능하다. 또한 블록 단위로 연산을 하기 때문에, 단일 segment 로 예측되는 블록에 대해서, Vincent-Soille 알고리즘의 연산 중 90% 이상을 차지하는 flooding 연산 부분을 수행하지 않도록 하는 것이 가능해졌다. 위의 방법을 multi-core system 에 적용하면, 2개의 processor 를 가지는 경우 약 1.9 배의 속도 향상이 가능하고, 4개의 processor 를 가지는 경우 약 3 배 정도의 속도 향상이 가능하다.

참고 문헌

[1] A.K. Jain, *Fundamentals of Digital Image Processing*, Prentice-Hall, 1989.

[2] M. Treiber, *An Introduction to Object Recognition: Selected Algorithms for a Wide Variety of Applications*, Springer, pp. 1-10, 145-147, 2010.

[3] K. Mikolajczyk, B. Leibe and B. Schiele, “Local Features for Object Class Recognition,” IEEE 10th International Conference on ICCV 2005, pp. 1792-1799, Oct. 2005.

[4] J.N. Wilson and G.X. Ritter, Handbook of Computer Vision Algorithms in Image Algebra, CRC Press, pp. 58-59, 2000.

[5] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond and D. Schmalstieg, "Pose Tracking from Natural Features on Mobile Phones," in Proc. of 7th IEEE/ACM ISMAR 2008, pp. 125-134, Sep. 2008.

[6] H.I. Koo and N.I. Cho, "Feature-based Image Registration Algorithm for Image Stitching Applications on Mobile Devices," IEEE Trans. on Consumer Electronics, vol. 57, no. 3, pp. 1303-1310, Aug. 2011.

[7] T. Tran and E. Marchand, “Real-Time Keypoints Matching: Application to Visual Servoing,” 2007 IEEE International Conference on Robotics and Automation, pp. 10-14, Apr. 2007.

[8] S. Battiato, G. Gallo, G. Puglisi and S. Scellato, "SIFT Features Tracking for Video Stabilization," 14th International Conference on Image Analysis and Processing 2007, pp. 10-14, Sep. 2007.

[9] D.G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” International Journal of Computer Vision, vol. 60, no. 2, pp. 91-110, Nov. 2004.

[10] T. Lindeberg, “Scale-Space for Discrete Signals,” IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 12, no. 3, pp. 234-254, Apr. 1990.

[11] M. Fischler and R. Bolles, “Random Sample Consensus: A Paradigm for Model Fitting with Application to Image Analysis and Automated Cartography,” Magazine on Communications of the ACM, vol. 24, no. 6, Jun. 1981.

[12] T. Tuytelaars and K. Mikolajczyk, "Local Invariant Feature Detectors: A Survey," Journal on Foundations and Trends® in Computer Graphics and Vision, vol. 3, no. 3, pp. 177-280, Jan. 2008.

[13] L. Juan and O. Gwun, "A Comparison of SIFT, PCA-SIFT and SURF," International Journal of Image Processing, vol. 3, no. 4, pp. 143-152, 2009.

[14] K. Yan, and R. Sukthankar, “PCA-SIFT: A More Distinctive Representation for Local Image Descriptors,” in Proc. of IEEE CVPR, pp. 506-513, Jul. 2004.

[15] H. Bay, A. Ess, T. Tuytelaars and L.V. Gool, "Speeded-Up Robust Features (SURF)," Elsevier Journal on Computer Vision and Image Understanding, vol. 110, no. 3, pp. 346-359, Jun. 2008.

[16] E.S. Kim and H.J. Lee, "A Novel Hardware Design for SIFT Generation with Reduced Memory Requirement," Journal of Semiconductor Technology and Science, vol. 13, no. 2, pp. 157-169, Apr. 2013.

[17] V. Bonato, E. Marques and G. A. Constantinides, “A Parallel Hardware Architecture for Scale and Rotation Invariant Feature Detection,” IEEE Trans. on Circuits and Systems for Video Technolory, vol. 18, no. 12, pp. 1703-1712, Dec. 2008.

[18] H.D. Chati, F. Muhlbauer, T. Braun and C. Bobda, “Hardware/Software co-design of a key point detector on FPGA,” in Proc. of IEEE Symposium FCCM, pp. 355-356, Apr. 2007.

[19] G. Michael, G. Helmut and B. Horst,“Fast Approximated SIFT,” in Proc. of ACCV 2006, pp. 918-927, Jan. 2006.

[20] K. Mikolajczyk and C. Schmid, ”A Performance Evaluation of Local Descriptors,” IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 27, no. 10, pp. 1615–1630, Oct. 2005.

[21] F.C. Huang, S.Y. Huang, J.W. Ker and Y.C. Chen, "High-Performance SIFT Hardware Accelerator for Real-Time Image Feature Extraction," IEEE Trans. on Circuits and Systems for Video Technology, vol. 22, no. 3, pp. 340-351, Mar. 2012.

[22] E. Rosten, R. Porter and T. Drummond, "Faster and better: A Machine Learning Approach to Corner Detection," IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 32, no. 1, pp. 105-119, Jan. 2010.

[23] A. Behrens and H. Rollinger, "Analysis of Feature Point Distributions for Fast Image Mosaicking Algorithms," Acta Polytechnica Journal of Advanced Engineering, vol. 50, no. 4, 2010.

[24] S. Gauglitz, L. Foschini, M. Turk and T. Höllerer, "Efficiently Selecting Spatially Distributed Keypoints for Visual Tracking," 18th IEEE International Conference on Image Processing 2011, pp. 11-14, Sep. 2011.

[25] M. Brown, R. Szeliski and S. Winder, "Multi-Image Matching Using Multi-Scale Oriented Patches," IEEE Computer Society Conference on CVPR 2005, pp. 510-517, Jun. 2005.

[26] G.P. Nguyen and H.J. Andersen, "Context-Based Adaptive Filtering of Interest Points in Image Retrieval," 9th International Conference on ISDA 2009, pp. 529-534, Dec. 2009.

[27] E. Ardizzone, A. Bruno and G. Mazzola, "Visual Saliency by Keypoints Distribution Analysis," Springer on Image Analysis and Processing–ICIAP 2011, pp. 691-699, Sep. 2011.

[28] Z. Cheng, D. Devarajan and R.J. Radke, "Determining Vision Graphs for Distributed Camera Networks Using Feature Digests," EURASIP Journal on Applied Signal Processing, vol. 2007, no. 1, Jan. 2007.

[29] V. Nannen and G. Oliver, "Grid-Based Spatial Keypoint Selection for Real Time Visual Odometry," in Proc. 2nd International Conference on Pattern Recognition Applications and Methods 2013, pp. 586-589, Feb. 2013.

[30] C. Gomila and F. Meyer, “Graph-based Object Tracking,” in Proc. of International Conference on ICIP 2003, pp. 14-17, Sep. 2003.

[31] L. Torresani, V. Kolmogorov and C. Rother, “Feature Correspondence Via Graph Matching: Models and Global Optimization,” in Proc. of Computer Vision–ECCV 2008, pp. 596-609, Oct. 2008.

[32] O. Duchenne, F. Bach, I. Kweon and J. Ponce, “A Tensor-Based Algorithm for High-Order Graph Matching,” IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 33, no. 12, pp. 2383-2395, Dec. 2011.

[33] M. Cho, J. Lee and K.M. Lee, “Feature Correspondence and Deformable Object Matching via Agglomerative Correspondence Clustering,” IEEE 12th International Conference on Computer Vision 2009, pp. 1280-1287, Oct. 2009.

[34] J.W. Jang and H.J. Lee, “Region-Constrained Feature Matching with Hierarchical Agglomerative Clustering,” VISAPP, 2014. (to be published)

[35] Xu Rui and D. Wunsch, "Survey of Clustering Algorithms," IEEE Trans. on Neural Networks, vol. 16, no. 3, pp. 645-678, May 2005.

[36] V. Ferrari, T. Tuytelaars and L.V. Gool, “Simultaneous Object Recognition and Segmentation from Single or Multiple Model Views,” International Journal of Computer Vision, vol. 67, no. 2, pp. 159-188, 2006.

[37] N. Abbas, “Graph Clustering: Complexity, Sequential and Parallel Algorithms”, Department of Computing Science, 1995.

[38] T. Hastie, R. Tibshirani and J. Friedman, “The Elements of Statistical Learning: Data Mining, Inference and Prediction,” Springer on Mathematical Intelligencer, vol. 27, no. 2, pp. 83-85, Jun. 2005.

[39] L. Vincent and P. Soille, “Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations,” IEEE Trans. on Pattern Analysis and Machine Intelligence, vol.13, no.6, pp. 583-598, Jun. 1991.

[40] J. Roerdink and A. Meijster, “The Watershed Transform: Definitions, Algorithms and Parallelization Strategies,” Fundamenta Informaticae, vol. 41, no. 1-2, pp. 187-228, 2000.

[41] J.W. Jang, H. Kim and H.J. Lee, "Block-based Marker Controlled Watershed Transform Using Motion Information", ITC-CSCC 2012, Jul. 2012.

[42] A.N. Moga and M. Gabbouj, "Parallel Image Component Labelling with Watershed Transformation," IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 19, no. 5, pp. 441-450, May 1997.

[43] S.Y. Chien, Y.W. Huang, and L.G. Chen, “Predictive Watershed: A Fast Watershed Algorithm for Video Segmentation,” IEEE Trans. on Circuits and Systems for Video Technology, vol.13, no.5, pp. 453- 461, May 2003.

[44] J. Kim, H.J. Lee, T.H. Lee, M. Cho, and J.B. Lee, "Hardware/Software Partitioned Implementation of Real-time Object-oriented Camera for Arbitrary-shaped MPEG-4 Contents," in Proc. of IEEE/ACM/IFIP Workshop on Embedded Systems for Real Time Multimedia 2006, pp. 7-12, Oct. 2006.

[45] B.J. Mealy, “Scanning Order Dependencies in Watershed Transform”, Technical Report UCSC-CRL-02-37, Dec. 2002.

[46] D. Wang and C. Labit, “Morphological Spatio-Temporal Simplification for Video Image Segmentation,” Signal Processing: Image Communication, vol. 11, no. 2, pp. 161-170, Dec. 1997.

[47] T.H. Kim, K.M. Lee and S.U. Lee, “A Unified Probabilistic Approach to Feature Matching and Object Segmentation,” 20th International Conference on ICPR 2010, pp. 464-467, Aug. 2010.

Abstract

Due to a popular use of personal storage devices and surveillance systems, the amount of stored image data is rapidly increasing. Therefore, a demand for an image recognition system using stored image data has been increased in a variety of applications such as object classification, object recognition, object tracking and so on. However, its excessive computation makes it difficult to be processed in real time by a device with limited resource such as a mobile phone. In this paper, several methods are proposed to perform object recognition at high speed.

First, this paper presents an adaptive keypoint generation for SIFT hardware implementation. In the image recognition system using local feature, SIFT (Scale-Invariant Feature Transform) has been proven to be one of the most robust local feature descriptors for various image transformations. To speed up SIFT generation, it is important to reduce the number of keypoints. The previous method apply the same threshold value regardless of the image characteristics. This paper addresses this problem and proposes a block-based keypoint adjustment controlling the keypoint generation using the characteristics of the block. The proposed method is capable of maintaining the pipeline structure of SIFT hardware and can generate a meaningful keypoint distribution. Furthermore, it is possible to reduce the execution time.

Second, feature matching method based clustering requires a large amount of computation because it needs to consider all correspondences of the image. This paper proposes a region-constrained feature matching in which an image is segmented into small regions and feature correspondences are clustered inside each region. The proposed region-constrained clustering dramatically reduces the execution time while it achieves a similar matching accuracy.

Third, watershed transform is difficult to process in parallel because it sequentially uses the values in the image. The proposed algorithm is processed in a block-based manner such that an image is decomposed into blocks and each block is processed independently of the other blocks. The block-based watershed transform can further reduce the computation by examining the necessity of flooding operations for each block and avoiding them when they are not necessary. The proposed method provides the same segmentation accuracy.

Keywords: SIFT, FAST, Hardware, SoC, Feature matching, Segmentation

Student number: 2008-30240